REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
1 April 1995

3. REPORT TYPE AND DATES COVERED
Final; 1 June 1991 to 31 August 1994

4. TITLE AND SUBTITLE

Protocol Engineering for Broadband Networks

5. FUNDING NUMBERS

DAAL03-91-G-0093

6. AUTHOR(S)

Ming T. Liu

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Department of Computer and Information Science
The Ohio State University
2015 Neil Avenue
Columbus, OH 43210-1277

8. PERFORMING ORGANIZATION REPORT

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

ARo 28467.27-EL

11. SUPPLEMENTARY NOTES

The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The final report summarizes the research results obtained, covering five areas in protocol engineering and distributed systems. The five areas are 1) protocol conversion, 2) conformance testing, 3) multimedia protocols, 4) high-speed packet switches, and 5) distributed control. For each area of research, the problems under investigation are first described, and then the major results obtained are summarized. A total of 28 publications, including 4 Ph.D. dissertations, 6 journal publications and 18 publications in international conference proceedings, have been credited to the contract. In the appendix, six reprints that were not included in two previous progress reports, are included in the final report.

14. SUBJECT TERMS

Protocol engineering, ATM networks, multimedia protocols, protocol coversion, conformance testing, distributed contro

15. NUMBER OF PAGES
138 pages

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT
UNCLASSIFIED

18. SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

19. SECURITY CLASSIFICATION OF ABSTRACT
UNCLASSIFIED

20. LIMITATION OF ABSTRACT
UL

PROTOCOL ENGINEERING FOR BROADBAND NETWORKS

FINAL REPORT

Dr. Ming T. Liu

March 30, 1995

U.S. ARMY RESEARCH OFFICE

Proposal Number: 28467-EL
Funding Document: DAAL03-91-G-0093

The Ohio State University
Department of Computer and Information Science
2015 Neil Avenue
Columbus, OH 43210-1277
Tel: (614) 292-6552 or 292-7084
FAX: (614) 292-2911
E-mail: liu@cis.ohio-state.edu

APPROVED FOR PUBLIC RELEASE;
DISTRIBUTION UNLIMITED

1 Foreword

The final report summarizes the research results obtained under ARO/CECOM contract No. DAAL03-91-G-0093. The research covered five areas in protocol engineering and distributed systems: protocol conversion, conformance testing, multimedia protocols, high speed packet switches and distributed control. A total of 28 publications have been credited to the contract, including 4 Ph. D. dissertations, 6 journal publications and 18 publications in international conference proceedings. Included in the appendices are six reprints of publications that were not included in two previous progress reports.

The research group should like to thank Drs. William A. Sander (ARO) and Charles J. Graft (CECOM) for their support and leadership to make the research both worthwhile and rewarding.

Contents

4. Appendices

A. H. W. Jeng and M. T. Liu, "From Service Specification to Protocol Converter: A Synchronizing Transition Set Approach," accepted for publication and to appear in *Computer Networks and ISDN Systems.*

B. H. W. Jeng and M. T. Liu, "Protocol Conversion in Multimedia Networks: Simulation and Algorithm," *Simulation*, vol. 64, pp. 51–60, January 1995.

C. A. Elsaadany, M. Singhal, and M. T. Liu, "Priority Communication Schemes on Local Area Networks for Multimedia Traffic," *Proc. 19th IEEE Conf. Local Computer Networks*, pp. 372–379, October 1994.

D. M. T. Liu, H.-W. Jeng, and L. S. Koh, "Formal Description Techniques for Protocol Specification," *Proc. of ATR Int'l Workshop on Communications Software Engineering*, pp. 31–71, October 1994.

E. L. S. Koh and M. T. Liu, "Test Path Selection Based on Effective Domains," *Proc. 1994 IEEE INt'l Conf. on Network Protocols*, pp. 64–71, October 1994.

F. C. J. Wang, L. S. Koh, and M. T. Liu, "Protocol Validation Tools as Test Case Generators," *Proc. 7th IFIP Int'l Workshop on Protocol Test Systems*, pp. 149–164, November 1994.

2 Report Body

In this section, the research results are presented. The research covered five areas: protocol conversion, conformance testing, multimedia protocols, high-speed packet switches and distributed control. The problems that were investigated in the respective areas are first described in Section 2.1, then the major results of the research are summarized in Section 2.2. A list of publication credited to this grant and the participating personnel can be found in Section 2.3 and Section 2.4, respectively.

2.1 Statement of Problems

2.1.1 Protocol Conversion

Due to the competitive and proprietary nature of computer networking and the rapid advances in computers and communications, many diverse network architectures and communication protocols have been developed during the past two decades. As a result, it poses problems for various networks to communicate with one another. To provide interoperability for systems residing on two networks that employ different protocols, some special devices, called gateways or converters, are needed to ensure that the transition sequences of one protocol are correctly converted to the transition sequences of the other protocol. Currently, most of the protocol converters are constructed in an ad hoc manners [29, 30, 31, 32]. However, both economic and technical consideration demand simpler and more reliable converters, which ad hoc converters cannot provide. To achieve the goal, formal methods for protocol conversion are needed. The objective of the research is to develop formal methods for constructing protocol converters so that the converters are free of deadlock, unexpected reception, improper termination and channel overflow.

2.1.2 Conformance Testing

When a formal Description Technique (FDT) is used to formally specify a communication protocol, and if the corresponding translator of the FDT can generate the error-free machine executable code for the protocol, then protocol testing is no longer required. Unfortunately, an implementation of the protocol always consists of two parts, the machine independent part and the machine-dependent part. Although the machine-independent part can be mechanically translated into machine executable code, the machine-dependent part must be manually encoded by protocol implementors. As a result, errors may be introduced into the implementation. Conformance testing is thus needed to make sure that the implementation conforms to its specification. For conducting conformance testing, a set of test sequences, called a test suite, must be generated. Our research objectives are (1) to develop a test architecture that integrates design and testing process into a single development environment so that the generation of test sequences can be facilitated, and (2) to develop procedures that can generate test sequences from the protocol specification.

2.1.3 Multimedia Protocols

To make multimedia applications a reality in the near future, two important problems need to be solved first: how to provide timely transmission of multimedia data over networks, specifically, local area networks and how to present multimedia data as a whole to end users. Traditional local area networks are not quite suitable for transmission of multimedia data. Multimedia data integrate video, voice, image and text, which may have stringent delay requirement. However, traditional local area networks are designed to handle regular data which are bursty in nature but allow variable delay. If multimedia data are transmitted in local area networks using existing protocols, the time-critical data will have to compete with regular data and will suffer intolerable delay. In addition, even if timely transmission of multimedia data can be provided, there is another problem of media synchronization. Since different kinds of multimedia data, such as video and voice, have different characteristics and requirements, they must be transmitted through networks in several media streams. These data, however, must be presented as a whole to end users at receiving ends. Since these different media streams may experience different network jitter and data loss during transmission, schemes are needed to resolve the problems in order to achieve synchronous presentation of data in various multimedia streams at the destinations.

2.1.4 High-Speed Packet Switches

The recent rapid development in integrated broadband networks has been driven by the push from new applications and the pull from the advance of new technologies. To realize integrated broadband networks, new switching technologies to support a variety of communication services with a wide range of data rates are essential. Mainly due to its flexibility, fast packet switching has emerged as the most appropriate switching techniques for integrated broadband networks. One great challenge is the design of packet switching fabrics that can switch packets at speeds several orders of magnitude higher than that of the current packet switches. Although there is no general consensus of what the switching fabrics for the future networks should be, most of the recent proposals in the literature have been based on a switching principle that employs (a) high degree of parallelism (b) distributed control, and (c) hardwired routing. However, to be deployed on a large scale, switching fabrics should also be fault-tolerant and congestion-free, in addition to the three requirements mentioned above. Our first objective in this research is to propose new high-speed switching fabrics that can satisfy the above requirements. In addition, the existing analytical models is not adequate to evaluate the performance of high-speed switching fabrics under nonuniform traffic patterns. Our second objective in this research is to address the formulation, mathematical modeling, and performance evaluation for high-speed switching fabrics.

2.1.5 Distributed Control

In this area, our research focuses on two topics. The first topic is related to distributed rule monitoring in distributed active databases. Production rules provide an unifying mechanism for several traditional features in database management systems and for new functionalities sought after by potential database applications. With integrated production rules, a database is transformed from traditionally passive to active such that the database will take predefined actions without user intervention when the database moves into certain states. Processing every rule upon every update in the system places a tremendous performance requirement on the success of active databases. However, the availability of multiple sites in a distributed database offers an opportunity to process rules in parallel. In addition, the features offered by production rules are desirable to distributed databases as well. The main objective of this research is to investigate the problem of monitoring rules in a distributed active database such that rules can be successfully integrated into distributed databases.

The other topic is related to the scheduling policies of distributed operating systems. Recently, workstation-based distributed systems have increasingly replaced large, multiuser, stand-alone computers. Distributed scheduling can be used to coordinate several inexpensive workstations so that their combined computing power is able to compete with that of an expensive centralized computer. When a scheduler finds that a remote processor is a better execution site for some jobs than the local processor, it transfers the job to the remote processor. While transferring jobs from overloaded to underloaded nodes in a distributed system can potentially improve performance, the characteristics of job vary enormously, and not all jobs are equally advantageous to transfer. To maximize performance, the jobs to be transferred must be selected carefully. Thus, a good selection policy is very important. However, little research has addressed this issue. Devising such a policy is particularly challenging because deciding the suitability of transferring a job implies predicting the resource requirements of the job before it is actually executed, which is a very difficult task. Our objective in this direction is to propose an intelligent job selection policy that can takes both the characteristics of the job and the current system environment into account, and that can adapt to changing job characteristics and environments.

2.2 Summary of Results

2.2.1 Protocol Conversion

Our contribution to this part of the research can be classified into the conversion specification approach and the service specification approach. The former uses conversion specifications which describe the mapping of transition sequences between two protocols and the latter uses service specifications which describe the required services offered by the final composite protocol.

For the conversion specification approach, two transformation procedures [1] are proposed that transform the protocol conversion problem into either the protocol validation problem or the protocol synthesis problem, depending on whether a gateway converter or an end-node converter is needed. Then, the existing protocol synthesis algorithms and validation algorithms can be used to construct a protocol converter for a given conversion specification. By transforming a protocol conversion problem into a protocol validation or a synthesis problem, the method proposed can take advantage of the transformation and use existing protocol validation or synthesis algorithms to construct a protocol converter.

For the service specification approach, two algorithms [16, 9] are proposed to formally derive a protocol converter from its service specification. The existing methods in the literature [33, 34] consider only the safety property in deriving protocol converters from service specification. As a result, they only guarantee that a subset of the required services is provided by the final composite protocol. Consequently, an extra phase is needed to verify that the final protocol can indeed support the whole service specification. The work proposed by us in [16] suggests the use of the system graph to construct protocol converters such that the system graph can be used not only to derive a protocol converter, but also to verify against the required service specification. Furthermore, a more elegant solution, called Synchronization Transition Set approach [9], is later on proposed. This solution is motivated by our observation that the main problems of the existing methods are, indeed, due to the fact that the information of the protocol implementation was not taken into consideration during the converter construction process. For a given service specification, there may be a number of different implementations for a protocol, and since the goal of protocol conversion is to find a correct converter for the existing protocol implementation, the information on the existing protocol specification must be taken into consideration for the conversion to be correct. Based on this observation, a 5-step protocol construction algorithm is proposed first to derive the conversion service requirement, and then to compose the final converter using the information on the existing protocol implementation at a later step. The converter so constructed has the three most desirable properties of a correct protocol converter; namely, conformity property, liveness property, and transparency property. Moreover, the proposed algorithm does not need a validation phase at the end to ensure the correctness of the converter as long as the protocols being converted are by themselves correct and the given conversion service requirement and service specifications are also correct; i.e. free from deadlock and livelock. In addition, the algorithm has the capability to handle the conversion between sequences of transitions in different protocols. This capability is crucial when dealing with complex protocols in which the semantics of sequences of transitions/messages is different from that of a simple concatenation of the semantics of each original individual transition/message and the conversion can only be done in a unit of sequence of transitions/messages.

Another major contribution of this part of the research is that a more efficient conversion model, called compensation model, is proposed to replace the traditional intermediate converter model for performing protocol conversion on multimedia networks [10]. Under this protocol compensation model, the Synchronizing Transition Set algorithm, is extended and modified to accommodate the high speed and high connectivity of the multimedia networks. The modified algorithm still retained all the desirable strengths of the original algorithm.

2.2.2 Conformance Testing

For test architecture, a computer-aided protocol design methodology based on a single representation mechanism, the OPS5 production system [35], is proposed to formally model the specification, validation, implementation and test phases [14]. To provide for direct control and observation during the test phase, the proposed architecture has (1) a single high level representation mechanism to be used in both design phase and test phase; (2) a Test sequence Generator and state Monitor (TGM) that is added to the local environment. By using a single representation mechanism approach, protocol test sequence generation is formally represented in production rules by combining those related elements in the formal protocol specification with additional elements used for controllability and observability in the test process. With the improved observability, the state of the implementation can be directly verified by checking the implementation state recorded after the application of a transition; with the improved controllability, the status of the implementation can be set to the head state's status of the transition to be tested.

For test sequence generation, our contribution can be summarized into two major areas. The first area is on the test procedures for protocol specified in Finite State Machine (FSM). The second area is on the Extended Finite State Machine (EFSM) which is more effective than the FSM in describing complex protocols. Our results in EFSM is important since there is little research done in this area.

For protocols specified in FSM, our major contribution is to improve the test coverage and the length of test sequences based on the unique input/output (UIO) method [36]. UIO is the most popular test sequence generation method for FSM. It has been improved to have better fault coverage by the revised UIO method, called UIOv method [37]. However, the UIOv method always needs a complete verification part. We propose a method that needs only a minimal verification part. Our method [13] has the same applicability as the UIOv method. In addition, our method uses a simple test on a given specification to decide whether a verification part is needed for detecting transfer faults in the protocol implementation. Furthermore, if a verification part cannot be completely omitted for detecting transfer faults, our method can generate a minimal number of input/output sequences for a verification part. In addition, two new approaches [17] are proposed to generate minimal length test sequences by using multiple UIOs and segment overlap. The first approach does not verify the uniqueness of UIO sequences of states, but the second approach does. Among test sequence minimization methods, the two approaches have the best applicability, and our second method has the best fault coverage.

In addition, a new approach, called Transition State Pair (TSP), is also developed for testing protocols specified in FSM [21]. The TSP method has three advantages over the W method [38] and the Wp method [39]. First, the TSP method can derive a test sequence not only for any minimal and fully specified protocols but also for minimal and partial specified protocols. Second, the TSP method uses less time to derive test sequences. Third, it derives shorter test sequences.

For EFSM, our main results are to propose an axiomatic approach for generating test sequences. To construct test sequences for EFSM, one must have a way to trace the changes and dependencies of the data items. A technique used in program proving, called axiomatic semantics [40], is found to be useful tool for serving such a purpose. Axiomatic semantics provides a set of axioms that describes the general rules of how the status of a program, called assertions, is changed before and after a statement. Depending on how the axioms are written, one can monitor different properties of a program and use the properties for different purposes. It is found that one can design the axiom in such a way that after a program is evaluated, the resulting assertion contains a test sequence. Based on this idea, a test procedure is proposed [15]. An improved version that simplifies the axioms, assertions, and the algorithms is also proposed [5] so that the axiomatic approach can be easily applied to any statement set. To extend the power of the approach, a method that takes in a requirement as a guide to generate test sequences for checking requirement conformity is proposed [23]. Most of the test sequence generation methods check transfer errors which are syntactic in nature. By using the requirements, semantic can be introduced into conformity testing. This enables test sequences to be generated with meaningful test purposes.

In addition, since methods of generating test sequences for testing EFSM focus mainly on the data flow aspect of a specification, the control aspect of the specification is ignored. In order to enhance the power of test sequences for EFSM, a method, called effective domain for testing, is proposed [27] to introduce UIO check sequences into the testing of the data flow of a protocol specification modeled by EFSM. By using the concept of effective domain, one can reduce the overall length of test sequences needed for checking both control flow and data flow.

Furthermore, a more flexible method is also proposed [18] for generating test sequences for a given fault model. Currently, most of the test generation procedures appearing in the literature generate test sequences to detect a specific type of errors, called a fault model. Since a test method is designed for a fixed fault model, the ability of detecting errors in the implementation is limited. To detect other types of faults, a procedure that can generate test sequences based on a given fault model is proposed. This method has also been extended [20] to generating test sequences for protocol specification written in the Estelle [41] which is an ISO standard specification language.

To take advantage of the availability of the existing validation software, a method is also proposed [28] to transform the problem of test sequences generation based on a fault model to a protocol validation problem. Protocol validation is used to determine whether a protocol specification is correct. This area has been studied for years, and many validation tools can be used to generate test sequences. By doing so, we can use these validation tools for the purpose of test sequences generation.

2.2.3 Multimedia Protocols

Our contributions include two parts. First two priority schemes are introduced into local area networks [25]. They are bus-time multiplexing priority scheme and station multiplexing scheme. Second, a multimedia synchronization protocol is proposed [24] to cope with different delay jitter and loss rate in several media streams. It is in charge of ensuring that the data sent in different media streams at the same time will also arrive at the destinations at about the same time within an acceptable tolerance.

To provide timely delivery of multimedia data in traditional local area networks, two level of priorities, high priority (for multimedia or real-time traffic) and low priority (for regular data), are introduced for data transmission [25]. Since traditional local area networks do not support any priority scheme, we propose two priority schemes for implementation in them. The basic idea behind these schemes is to let the stations wishing to transmit multimedia data inform other stations to withhold from transmitting any regular data. Simulation results show that by incorporating these priority schemes into local area networks, response time for multimedia traffic is improved and packets discarding possibilities for multimedia data packets are reduced.

The multimedia synchronization protocol we proposed [24] serves as an interface between the underlying ATM network and the multimedia applications. It consist of two entities, the marker, which insert some control cells, called sync cells, into the media streams to mark the places where data should be synchronized, and the synchronizer, which recognizes the control cells and align the data delivery in different media streams. To cope with the loss of sync cell, three policies, drop-old, transmit-old and delayed-transmit, are considered for implementation within the synchronization protocol. For these three policies, the translation from the quality of service (QOS) specified by the multimedia applications into the QOS for the ATM network has been analyzed. According to these analysis, it can be deduced that the drop-old policy is more demanding on network cell loss rate. However, the drop-old policy is less restrictive on network delay requirement. The delayed-transmit policy imposes stronger demand on network delay, but it allows a smaller portion of cells to be received correctly in that delay period. In addition, it requires a smaller buffer size.

2.2.4 High-Speed Packet Switches

Our main contributions include two parts. First, a new indirect star-type network, called the μ-star network [6], is proposed. The μ-star network is an indirect star network. It is obtained by an unfolding scheme applied to the star graph based on its recursive property. Being a star-type network, the proposed μ-star network inherits all the good properties from the star graph. The star graph has a smaller degree and diameter than the well-known n-cube, and can be an alternative to the n-cube for systems with a large number of nodes. It also is symmetric, highly modular, strongly hierarchical, and maximally fault-tolerant [42, 43, 44, 45], like n-cube type network. Because of its symmetry, the star graph is easily extensible and can be decomposed in many different ways. Its routing algorithm is also very simple. The star graph is Hamiltonian. Efficient sorting algorithms, a collection of application algorithms and a parallel algorithm for computing fast Fourier transforms on the star graph are available [46, 47, 48, 49].

The μ-star network is modular. Therefore, a large μ-star network can be built from smaller ones. The performance of the μ-star network has been analyzed under the uniform traffic model and shown to be almost identical to that of the indirect cube-type network based on (2×2) switches. Its performance also shows that it is an improvement over previously proposed indirect star-type networks [50]. The routing of the μ-star network is simple and the routing decision can be made locally within each of the switches in the network. The μ-star network can be an alternative to the indirect cube-type network when the network size is close to a factorial rather than a power of a integer.

Our second contribution is to present a new, and more general analytic model [8] for the Knockout Switch. The Knockout Switch is a nonblocking, high performance switch suitable for broadband packet switching. It allows packet losses, but the probability of a packet loss can be kept extremely small in a cost-effective way. The performance of the Knockout Switch was analyzed under uniform traffic [51], and it shows that the Knockout Switch does have extremely low packet loss probabilities under uniform traffic. In this part of research, we proposed a more general analytical model to analyze the lost packet probabilities and the effectiveness of the Knockout Switch under various nonuniform traffic patterns. This analytic model is an approximate Markov chain. It integrates the effects of the concentrator and the shared buffer on the lost packet probability, and allows any traffic patterns.

The accuracy of the new analytic model is verified by comparing the numeric results obtained from the new model to the known results for uniform traffic in [51]. This research shows that the (1024 × 1024) Knockout Switch is stable under hot-spot traffic with the buffer size of 40 and the threshold of 8, if the hot-spot load is limited to keep the lost packet probability $\leq 10^{-6}$. Once the hot-spot load becomes higher than these cut-off points, the lost packet probability becomes higher than 10^{-6}. At this point, increased buffering and threshold do not help. We observed the same phenomenon for smaller network sizes, too. This suggests the importance of regulating the hot-spot load within the range not to overload the hot-spot to keep the Knockout Switch stable. Under the mixed point-to-point/uniform traffic that has particular significance of mixed voice, data, and video applications, the analytical result shows that the Knockout Switch is stable given sufficient buffers. The research result also shows that Knockout Switch performs stably under two other nonuniform traffic patterns, pattern I and pattern II. On the contrary, it has been shown in [52] that the multistage packet switches, such as Banyan networks, significantly degrades their performance under these traffic models. In addition, the research result shows that the buffer size needs to be increased from 40 to 80 to keep the lost packet probabilities under 10^{-6} under the point-to-point traffic and traffic pattern I.

2.2.5 Distributed Control

In the area of distributed databases, our major contribution is that an integrated approach, including a rule decomposition scheme, a distributed algorithm and a distributed evaluation algorithm, is used for processing complicated rules in distributed active databases [12, 7, 22]. The decomposition scheme uses a new relational operator, AND, to identify independent parts of a rule query and to facilitate the distribution of subrules. The distribution of the subrules will adapt to the relation distribution, which has not been investigated before. The distributed evaluation algorithm makes use of the different types of nodes derived from the AND operator to collect and combine local results to derive a consistent global rule evaluation result. A consistent global evaluation result is difficult to achieve due to the distributed environment and has not been addressed previously in distributed rule processing. The concept of simultaneous regions is applied to distributed rule evaluation to achieve consistency and correctness. The performance analysis of the distributed evaluation algorithm suggests that fatter trees are preferable to taller trees in terms of the response time and the message count.

In the area of distributed operating systems, we proposed an intelligent job selection policy that learns the behavior of the wide variety of job types that make up the workload of a typical distributed system [19]. A learning mechanism, called weight climbing, is used to gather knowledge of how each type of job behaves. Based on this knowledge, our selection policy decides which jobs can be advantageously transferred under the current conditions. Such a job scheduler has the following advantages: first, the job transfer decision is made based not only on current conditions, but on detailed characteristics of the specific job being considered. Second, knowledge of these characteristics is accurate, because it is acquired through continuing observation on the particular system in which transfer is being considered. Third, whenever the system configuration or workload changes, the scheduler can relearn the situation and adapt itself automatically. Our experimental results show that our intelligent selection policy is able to learn job behavior quickly and to make decisions accordingly. The performance of our policy is compared with that of the Threshold policy [53]. The results show that, as our policy learns job behaviors, it is able to significantly improve performance relative to the Threshold policy. The results also show that our policy automatically adapts to system configuration or program behavior changes. The job selection policy we proposed not only takes both the characteristics of the jobs and the current system environment into account, but also adapts to changing job characteristics and environments. These features are provided with negligible time and space overhead.

2.3 Publications

The following is a list of 28 publications that have been credited to the grant.

2.3.1 Ph. D. Dissertations (4)

[1] J. Chang, *Transformation Approaches to Protocol Internetworking.* PhD thesis, Dept. of Computer and Information Science, The Ohio State University, March 1992.

[2] I. M. Hsu, *Distributed Rule Monitoring in Distributed Active Databases.* PhD thesis, Dept. of Computer and Information Science, The Ohio State University, March 1993.

[3] S. S. Yu, *Test Sequence Generation Methods for Communication Protocols.* PhD thesis, Dept. of Computer and Information Science, The Ohio State University, August 1993.

[4] C. J. Wang, *Automatic Test Case Generation of Conformance Testing for Communication Protocols Specified in Extended Models.* PhD thesis, Dept. of Computer and Information Science, The Ohio State University, June 1994.

2.3.2 Journal Publications (6)

[5] C.-J. Wang and M. T. Liu, "A Test Suite Generation Method For Extended Finite State Machines Using Axiomatic Semantics Approach," *IFIP Trans. Protocol Specification, Testing, and Verification, XII*, pp. 29–43, North-Holland, 1992.

[6] W. S. Chen, K. Y. Lee, and M. T. Liu, "μ-Star: A Modular Indirect Star Network," *Int'l Journal of Networks*, vol. 23, pp. 261–270, July 1993.

[7] I. M. Hsu, M. Singhal, and M. T. Liu, "Distributed Rule Monitoring in Active Databases," *Integrated Computer-Aided Engineering*, vol. 1, no. 4, pp. 295–309, 1994.

[8] H. Yoon, M. T. Liu, and Y. M. Kim, "The Knockout Switch Under Nonuniform Traffic," accepted for publication and to appear in *IEEE Trans. on Communications*, 1995.

[9] H. W. Jeng and M. T. Liu, "From Service Specification to Protocol Converter: A Synchronizing Transition Set Approach," accepted for publication and to appear in *Computer Networks and ISDN Systems.*

[10] H. W. Jeng and M. T. Liu, "Protocol Conversion in Multimedia Networks: Simulation and Algorithm," *Simulation*, vol. 64, pp. 51–60, January 1995.

2.3.3 Publications in Conference Proceedings (18)

[11] S. S. Yu and M. T. Liu, "Using Heuristic to Guide Reachability Analysis," *Proc. SICON'91*, pp. 226–231, September 1991.

[12] I. M. Hsu, M. Singhal, and M. T. Liu, "Distributed Rule Processing in Active Databases," in *Proc. 8th Int'l Conf. on Data Engineering*, pp. 106–113, February 1992.

[13] S. S. Yu and M. T. Liu, "A New Protocol Test Sequence Generation Methods Based on UIOS," *Proc. INFOCOM'92*, pp. 2068–2077, May 1992.

[14] C. M. Huang and M. T. Liu, "An Incremental Protocol Test Method: Formal Modelling and Architectures," in *Proc. 2nd Int'l Conf. on Systems Integration*, pp. 499–508, June 1992.

[15] C.-J. Wang and M. T. Liu, "Axiomatic Test Sequence Generation for Extended Finite State Machines," in *Proc. 12th International Conference on Distributed Computing Systems*, pp. 252–259, June 1992.

[16] Y. W. Yao and M. T. Liu, "Constructing Protocol Converters from Service Specifications," in *Proc. 12th International Conference on Distributed Computing Systems*, pp. 344–351, June 1992.

[17] S. S. Yu and M. T. Liu, "Utilizing Multiple UIO Sequences and Segment Overlap to Shorten Test Sequences," *Proc. ICCS/ISITA'92*, pp. 1291–1296, November 1992.

[18] C.-J. Wang and M. T. Liu, "Generating Test Cases for EFSM with Given Fault Models," in *Proc. IEEE INFOCOM '93*, pp. 774–781, March 1993.

[19] C. J. Wang, P. Krueger, and M. T. Liu, "Intelligent Job Selection for Distributed Scheduling," in *Proc. 13th International Conference on Distributed Computing Systems*, pp. 517–524, June 1993.

[20] C. J. Wang and M. T. Liu, "Automatic Test Case Generation for Estelle," in *Proc. of 1993 International Conference on Network Protocols*, pp. 774–781, October 1993.

[21] S. S. Yu and M. T. Liu, "The Transition-State Pair Method for Test Sequence Generation," in *Proc. IEEE Globecom'93*, pp. 1029–1033, November 1993.

[22] I. M. Hsu, M. Singhal, and M. T. Liu, "Performance Study of Distributed Rule Evaluation Algorithm in Distributed Active Databases," in *Proc. 13th IEEE Annual Int'l Phoenix Conf. on Computers and Communications*, pp. 24–30, April 1994.

[23] L. S. Koh, C. J. Wang, and M. T. Liu, "A Functional Model for Test Sequence Generation," in *Proc. 13th IEEE Annual Int'l Phoenix Conf. on Computers and Communications*, pp. 336–342, April 1994.

[24] C. J. Wang, L. S. Koh, C. H. Wu, and M. T. Liu, "A Multimedia Synchronization Protocol for ATM Networks," in *Proc. 14th International Conference on Distributed Computing Systems*, pp. 476–483, June 1994.

[25] A. Elsaadany, M. Singhal, and M. T. Liu, "Priority Communication Schemes on Local Area Networks for Multimedia Traffic," in *Proc. 19th IEEE Conf. Local Computer Networks*, pp. 372–379, October 1994.

[26] M. T. Liu, H.-W. Jeng, and L. S. Koh, "Formal Description Techniques for Protocol Specification," in *Proc. of ATR Int'l Workshop on Communications Software Engineering*, pp. 31–71, October 1994.

[27] L. S. Koh and M. T. Liu, "Test Path Selection Based on Effective Domains," in *Proc. 1994 IEEE INt'l Conf. on Network Protocols*, pp. 64–71, October 1994.

[28] C. J. Wang, L. S. Koh, and M. T. Liu, "Protocol Validation Tools as Test Case Generators," in *Proc. 7th IFIP Int'l Workshop on Protocol Test Systems*, pp. 149–164, November 1994.

2.4 Participating Personnel

In addition to Principal Investigator, Dr Ming T. Liu, the following Graduate Research Associates participated in the research project:

- Ms. Jing Chang (Ph. D., March 1992)
- Ms. Ing-Miin Hsu (Ph. D., March 1993)
- Ms. Sarah S. Yu (Ph. D., March 1994)
- Chang-Jia Wang (Ph. D., June 1994)
- Hou-Wa Jeng (Ph. D., June 1995, expected)
- Amr Elsaadany (Ph. D., June 1995, expected)
- Liang-Seng Koh (Ph. D., 1996 expected)
- Ms. Chao-Hui Wu (Ph. D., 1996 expected)

3 Bibliography

[29] P. Francois and A. Potocki, "Some Methods for Providing OSI Transport in SNA," *IBM Journal of Research & Development*, vol. 27, pp. 452–463, Sept. 1983.

[30] I. Groenback, "Conversion Between the TCP and ISO Transport Protocols as a Method of Achieving Interoperability Between Data Communication Systems," *IEEE JSAC*, vol. SAC-4, pp. 288–296, Mar. 1986.

[31] J. M. Rodriguez, "An X.PC/TCP Protocol Translator," in *Proc. IEEE INFOCOM 1988*, pp. 308–313, Mar. 1988.

[32] M. T. Rose and D. E. Cass, "OSI Transport Service on Top of TCP," *Computer Networks and ISDN Systems*, vol. 12, pp. 159–173, 1987.

[33] K. L. Calvert and S. S. Lam, "Deriving a Protocol Converter: A Top-Down Method," in *Proc. ACM SIGCOMM 1989 Symposium*, (Austin), pp. 247–258, Sept. 1989.

[34] K. Okumura, "Generation of Proper Adaptors and Converters from a Formal Service Specification," in *Proc. IEEE INFOCOM 1990*, (San Francisco), pp. 564–571, Jan. 1990.

[35] L. Brownston, R. Farrel, E. Kart, and N. Martin in *Expert Systems in OPS5*, Addison-Wesley Publishing Company, 1985.

[36] K. Sabnani and A. Dahbura, "A Protocol Test Generation Procedure," *Computer Networks and ISDN Systems*, vol. 15, pp. 285–297, 1988.

[37] W. Y. L. Chan, S. T. Vuong, and M. R. Ito, "An Improved Protocol Test Generation Procedure Based on UIOs," in *Proc. ACM SIGCOMM '89 Symp.*, pp. 283–294, 1989.

[38] T. Chow, "Testing Software Design Modeled by Finite-State Machines," *IEEE Trans. on Software Engineering*, vol. SE-4, pp. 178–187, March 1978.

[39] S. Fujiwara, G. v. Bochmann., F. Khendek, M. Amalou, and A. Ghedamsi, "Test Selection Based on Finite State Models," *IEEE Trans. Software Engineering*, vol. SE-17, pp. 591–603, June 1991.

[40] F. G. Pagan, *Formal Specification of Programming Languages: A Panoramic Primer*, ch. 4, pp. 193–215. Prentice-Hall, Inc., 1981.

[41] S. Budkowski and P. Dembinski, "An Introduction to Estelle: A Specification Language for Distributed Systems," *Computer Networks and ISDN Systems*, vol. 14, no. 1, pp. 3–23, 1987.

[42] S. B. Akers and B. Krishnamurthy, "A Group Theoretic Model for Symmetric Interconnection Networks," *Proc. of International Conference on Parallel Processing*, pp. 216–223, August 1986.

[43] S. B. Akers, D. Harel, and B. Krishnamurthy, "The Star Graph: An Attractive Alternative to the n-Cube," *Proc. of International Conference on Parallel Processing*, pp. 393–400, August 1987.

[44] S. B. Akers and B. Krishnamurthy, "The Fault Tolerance of Star Graphs," *Proc. of the 2nd International Conference on Supercomputing*, 1987.

[45] S. B. Akers and B. Krishnamurthy, "A Group Theoratic Model for Symmetric Interconnection Networks," *IEEE Trans. on Communications*, vol. 38, pp. 555–566, April 1989.

[46] P. Fragopoulou and S. G. Akl, "A Parallel Algorithm for Computing Fourier Transforms on the Star Graph.," technical report, Queen's University, Kingston, Ontario, Canada, October 1990.

[47] A. Menn and A. K. Somani, "An Efficient Sorting Algorithm for the Star Graph Interconnection Network," *Proc. of the 1990 International Conference on Parallel Processing*, pp. 1–8, August 1990.

[48] M. Nigam, S. Sahni, and B. Krishnamurthy, "Embedding Hamiltonians and Hypercubes in Star International Graphs," *Proc. of the 1990 International Conference on Parallel Processing*, pp. 1–8, August 1990.

[49] K. Qiu, H. Meijer, and S. G. Akl, "Parallel Routing and Sorting on the Packet Network," *Proc. of International Conference on Computing and Information*, pp. 360–371, May 1991.

[50] K. Y. Lee and H. Yoon, "Indirect Star-Type Networks for Large Multiprocessor Systems," *IEEE Transactions on Computers*, vol. 31, pp. 1277–1282, Novermber 1991.

[51] U. Yeh, M. G. Hluchyj, and A. S. Acampora, "The Knockout Switch: A Simple Modular Architecture for High-Performance Packet Switching," *IEEE Journal on Selected Areas in Communications*, vol. 5, no. 8, pp. 1274–1283, 1987.

[52] L. T. Wu, "Mixing Traffic in a Buffered Banyan Network," *Proc. of ACM 9th Data Communication Symposium*, pp. 134–139, 1985.

[53] D. L. Eager, E. D. Lazowska, and J. Zahorjan, "A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing," *Performance Evaluation*, vol. 6, no. 1, pp. 53–68, 1986.

A. **H. W. Jeng and M. T. Liu,**
"From Service Specification to Protocol Converter: A Synchronizing Transition Set Approach,"
accepted for publication and to appear in *Computer Networks and ISDN Systems.*

From Service Specification to Protocol Converter: A Synchronizing Transition Set Approach*

Hou-Wa J. Jeng, and Ming T. Liu
Department of Computer and Information Science
The Ohio State University
2036 Neil Avenue
Columbus, OH 43210-1277

Abstract

With the proliferation of different network architectures, it has been recognized that protocol conversion is needed for achieving the interoperability between computer networks that implement different protocols [1, 2, 3]. Since 1986, many protocol converter construction algorithms based on formal models have been proposed. In the meantime, it has been observed by many researchers that algorithms based on service specification have in general less complexity at the design stage. In this paper, a five-step algorithm is proposed to formally derive a protocol converter from the service specification: (1) Generate the *Service System Graph* from the given service specifications; (2) Derive the *Conversion Service Specification* from the service system graph; (3) Generate the *Synchronizing Transition Sets* from the conversion service specification; (4) Compose the converter using the synchronizing transition sets; and (5) Remove the remaining service transitions to obtain the final *Protocol Converter Specification*. The converter so constructed has the three most desirable properties of a correct protocol converter; namely, conformity property (also known as maximum safety property), liveness property, and transparency property. Moreover, unlike many other protocol conversion algorithms, the proposed algorithm does not need a validation phase at the end to ensure the correctness of the converter as long as the protocols being converted are by themselves correct and the given conversion service requirement and service specifications are also correct; i.e. free from deadlock and livelock. The reason is that the proposed algorithm preserves all the properties and functionalities of the original protocols during the derivation of the converter. In addition, it has the capability to handle the conversion between sequences of transitions in different protocols. This capability is crucial when dealing with complex protocols in which the semantics of sequences of transitions/messages is different from that of a simple concatenation of the semantics of each original individual transition/message and the conversion can only be done in a unit of sequence of transitions/messages. For ease in specification and discussion, the formal CFSM (Communication Finite State Machine) model is adopted in this paper. Nevertheless, the same algorithm can be easily extended to work effectively in the Extended Finite State Machine (EFSM) model as well [4].

Keywords: synchronizing transition set, protocol conversion, protocol converter, protocol specification, service specification

1 Introduction

With the development of a variety of networks in recent years, internetworking between different networks has become an important issue since users on differ-

[1] Research reported herein was supported by U.S. Army Research Office, under contract Nos. DAAL03-91-G-0093 and DAA03-92-G-0184. The views, opinions, and/or findings contained in this paper are those of the authors and should not be construed as an official Department of the Army position, policy or decision.

ent networks need to communicate with each other. Due to the so called *protocol mismatches* [1], protocol conversion has become a necessity for internetwork communication in a heterogeneous network environment. In [1], Green analyzed different internetwork architectures and situations in which conversion has to be performed. Even in an ideal case in which the convergence into worldwide standard protocols may reduce the need of conversion, it is still far away from being practical, simply because of the high cost and other reasons. In fact, recent development in the worldwide ISDN (Integrated Services Digital Network) adopts an evolutionary instead of a revolutionary approach; this trend further indicates the necessity of protocol conversion. Consequently, the conversion problem, arising from the need for constructing a converter that provides interoperability between different network protocols, has become an important design issue in computer communication.

During the past decade, a number of conversion algorithms have been proposed. Many of them construct converters in an ad hoc manner [5, 6, 7, 8]. On the other hand, following Green's pioneering work in [1], many algorithms based on formal models have also been proposed. It is believed that to solve the protocol conversion problem efficiently and once for all, the formal method is the right direction to follow. In this paper, a five-step conversion algorithm based on the CFSM (Communication Finite State Machine) model is proposed to formally derive a protocol converter from a service specification.

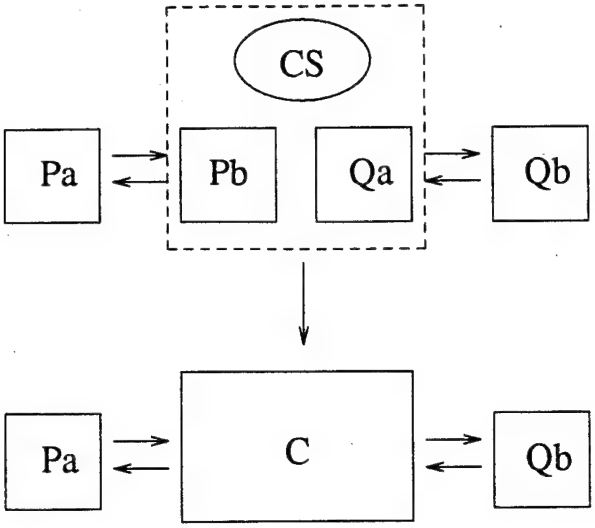
Among those formal methods proposed by different researchers, there are in general two different approaches: *Conversion Specification* and *Service Specification*. To derive a protocol converter, the conversion specification approach takes directly as input the original protocol specifications and the given conversion specification, which is in a similar form as the protocol specification, as illustrated in Fig. 1. Essentially, the conversion specification either tells how to translate the messages between protocols or specifies the ordering of the given protocol transitions to synchronize the execution of the given protocol entities. In other words, the conversion specification tells how to perform the protocol conversion for the given protocols. The algorithms proposed in [9, 10, 11, 12, 13, 14, 15, 16] belong to this category.

One of the shortcomings of the conversion specification approach is that the implementation details are involved at the early stage of design, thereby resulting in a more complex algorithm. In addition, there is no formal algorithm for generating the conversion specification, which is usually obtained in an ad hoc manner. This is a very serious drawback, especially for those complex protocols whose conversion specifications are hard to find in ad hoc ways.

On the other hand, the service specification approach, which has received more and more attention in recent years, treats the implementation details as unknown *black boxes* at the early stage of design; it only depicts the final behavior of the overall protocol system as a to-be-found converter from the viewpoint of the protocol service users at the *SAP* (Service Access Point). The service specification is specified in terms of the *Service Transitions* to be executed at the SAPs. Fig. 2 shows the high level concept of the service specification approach. The algorithms proposed in [17, 18, 19, 20, 21] belong to this category.

Due to its high level of abstraction, finding a correct service specification at the protocol service user level is easier and more practical than finding the conversion specification in an ad hoc manner by examining the implementation of the given protocols. However, the difficulty of the service specification approach is, in general, that the service specification by itself does not tell directly how to perform the protocol conversion. As a result, the *information* on how to perform the conversion must be derived from the given service specifications and requirement. Since the conversion information derived is not specific, the resulting converter may not be correct. Consequently, most of the algorithms in this category require a validation phase at the end to ensure the correctness of the protocol converter constructed. This not only complicates the construction procedure but also means that when the validation fails, the procedure has failed to find a converter for the given service specifications and requirement. Moreover, even when the validation phase may pass, the converter so constructed often covers only a subset of the properties and functionalities of the original protocols [3, 21]. This drawback, as was observed, is due to the fact that the information on the existing protocol implementation was not taken into consideration during the converter construction process.

Due to the fact that for a given service specification there may be a number of different implementations of a protocol (as illustrated in the example in Section 3 of this paper), and the goal of the protocol conversion algorithm is to find a correct converter for the existing protocol implementation, the information on the the existing protocol specification (implementation) must be taken into consideration for the conversion to be



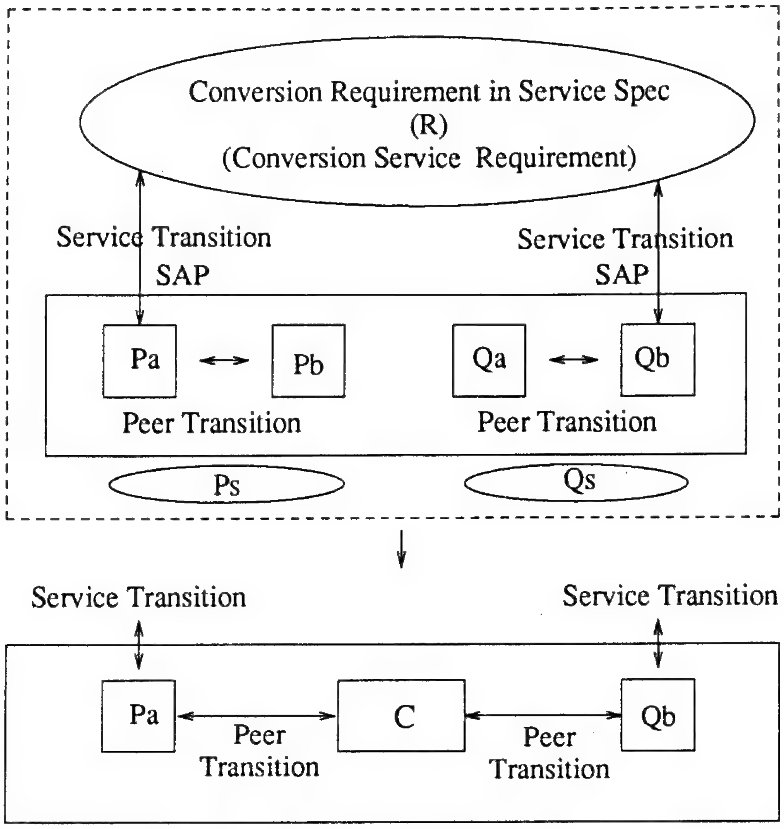
P_a, P_b, Q_a and Q_b: protocol specifications
CS: conversion specification
C: final converter specification

Figure 1: The Conversion Specification Approach

correct. In other words, the information contained in the service specifications alone is not sufficient to derive a correct converter, and different protocol implementations of the same service specification will result in different protocol converter specifications.

In the above, it was claimed that the information on the protocol implementation must be taken into consideration in the protocol derivation procedure. It was also mentioned that to take advantage of the low-complexity in the service specification approach, the implementation details should not be involved too early in the derivation procedure. In this paper, a 5-step protocol converter construction algorithm is proposed first to derive the conversion service specification from the given conversion service requirement, and then to compose the final converter using the information on the existing protocol implementation at a later step (Step 4). The algorithm will always find a *maximum* converter that meets the requirement as specified in the conversion service requirement, and the resulting converter also always has the three most desirable properties of a correct protocol converter: conformity property (also known as maximum safety property), liveness property, and transparency property. The conformity property means that the converter so constructed supports no more and no less functions than the original protocols do. The liveness property means that the converter is free from deadlock and livelock as long as the protocols being converted and the conversion service requirement given have the liveness property by themselves. The transparency property means that the conversion algorithm does not alter the protocol entities at the end node of the composed protocol system and the conversion is transparent to the end users.

Another important capability of the proposed algorithm is that it is able to handle the conversion between sequences of transitions in different protocols. This is a very encouraging result because the semantics of a sequence of transitions/messages can be different from that of a simple concatenation of the semantics of each original individual transition/message for complex protocols, and because many of the existing conversion algorithms can only handle mapping between single transition/message. In other words, when simple one-to-one transition/message mapping between protocols is not sufficient to derive a correct converter and the conversion can only be done in a unit of sequence of transitions/messages, the proposed algorithm can still derive a correct converter



P_a, P_b, Q_a and Q_b: protocol specifications
P_s, Q_s: service specifications
R : Conversion Service Requirement
C: final converter specification

Figure 2: The Service Specification Approach

succeesfully. This capability of performing the conversion between sequences of transitions in different protocols is demonstrated in Example 2 (in Section 4) of this paper.

The rest of the paper is organized as follows: Section 2 describes the model and the specification method used; Section 3 and Section 4 give examples to describe the proposed algorithm, and discuss the ideas behind each step of the algorithm and the strategies that can be used to reduce the complexity. Section 5 provides a formal proof of correctness to show the reasons why an extra validation step is not needed for the algorithm. Finally, Section 6 concludes the paper. For easy reference, an outline of the algorithm is provided in Appendix.

2 Models

In this section the model and specification method adopted are described and the protocol conversion problem is formally defined in terms of the specification method used.

A class of state-transition systems, called CFSMs (Communication Finite State Machines), is adopted for the protocol specification and service specification. A protocol system consists of a set of CFSMs called *protocol entities*. Each protocol entity communicates with the others through message passing. For simplicity only 2-entity protocols are considered in this paper. The model is shown in Fig. 3.

The communication channel between the entities

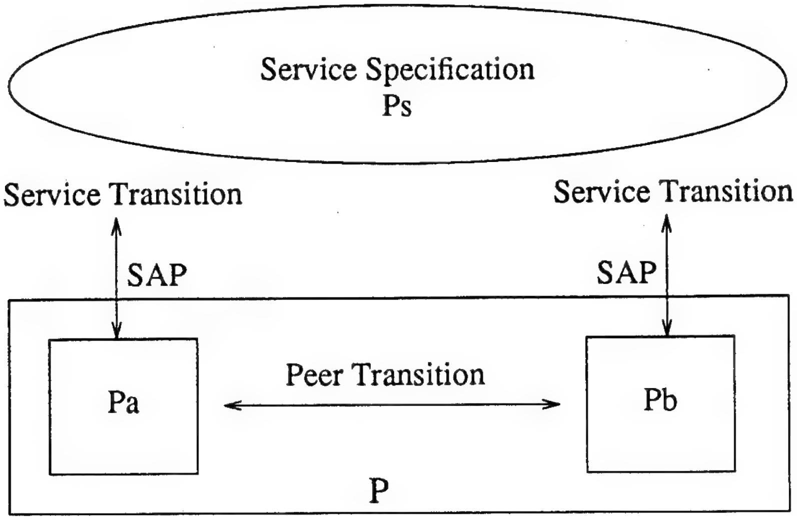


Figure 3: A Protocol System in CFSM Model

can be either reliable or noisy. If the channel between protocol entities P_a and P_b is noisy, then the given protocol P must be able to handle the noisy situation. As far as the conversion is concerned, the proposed algorithm covers both cases since the functionalities of the original protocol are to be preserved. In the model, however, a distinction is made between two kinds of transitions executed in the entities, namely, *Peer Transitions* that support the interaction between two peer protocol entities and *Service Transitions* that perform the interaction between a protocol entity and its environment (e.g. the protocol service users) through the SAP (Service Access Point). The distinction is made because of the observation that there are no service users on top of the final converter as depicted in Fig. 2 and all the service transitions generated during the construction process will be removed from the final converter. Formally, a CFSM protocol entity is defined as follows.

Definition 1. A CFSM Protocol Entity P_x is a six-tuple $(q, \Sigma, \lambda, \delta, i, f)$, where:

1. q is the set of states in P_x.
2. Σ is the finite set of service transitions in P_x.
3. λ is the finite set of peer transitions in P_x.
4. δ is the transition function of P_x which maps $q \times (\Sigma \cup \lambda)$ into q; if $q(s_1, t) = s_2$, where $s_1 \in q, s_2 \in q$, and $t \in (\Sigma \cup \lambda)$, s_1 is called the starting state of t and s_2 is the ending state of t. Note that a *receive/read* transition t_r is said to be executable, only when the current state of P_x is the starting state of t_r and the *message to be received/read* is already in the incoming channel to P_x.
5. i is the initial state of P_x.
6. f is the set of the final states in P_x. Note that for a typical non-terminating protocol, f usually has only one element, i, which is the initial state of P_x

For ease in discussion it is assumed in this paper that all the protocol entities are deterministic finite state machines, and that the *protocol specification* consists of a set of CFSM protocol entities so defined. The formal definition of the *Service Specification* can be defined similarly as follows.

Definition 2. A CFSM Protocol Service Specification P_s, $(q, \Sigma, \lambda, \delta, i, f)$, is a six-tuple, where:

1. q is the set of states in P_s.
2. Σ is the finite set of service transitions in P_s, which is the union of the service transitions of all the protocol entities.

3. λ is the finite set of peer transitions in P_s and it is an empty set.
4. δ is the transition function of P_s, which maps $q \times \Sigma$ into q; if $q(s_1, t) = s_2$, where $s_1 \in q, s_2 \in q$, and t $\in \Sigma$, s_1 is called the starting state of t and s_2 is the ending state of t.
5. i is the initial state of P_s.
6. f is the set of the final states in P_s. Again, for a typical non-stopping protocol, f usually has only one element, i, which is the initial state of P_s.

As illustrated in Fig. 3, the service specification describes how the protocol system functions from the viewpoint of the service users. It does not specify how the peer entities interact with each other. In terms of the service specification, the *Protocol Conversion Problem* is formally defined as follows.

Definition 3. Protocol Conversion Problem.

Given:

1. The CFSM protocol entities P_a, P_b, Q_a, Q_b and the CFSM service specifications P_s and Q_s.
2. The Conversion Service Requirement, R, which describes how the composed protocol system should function. R is specified in the same form as the CFSM service specification and the transition set of R is the union of the service transition sets of P_a and Q_b. i.e.

$$\Sigma_R = \Sigma_{P_a} \cup \Sigma_{Q_b}$$

Goal:
Obtain a converter, C, as illustrated in Fig. 2. The converter is specified in the same form as a CFSM protocol entity and its service transition set is empty. In addition, C must have the following properties for correctness.

1. Conformity Property: C should support no more and no less functions than the original protocols do, and the final protocol system with the constructed converter must conform to the *conversion service requirement, R.*
2. Liveness Property: C should be free from deadlock and livelock if P, Q and R are also free from deadlock and livelock.
3. Transparency Property: P_a and Q_b must be preserved and can communicate with each other through C.

Note that a more restricted definition of a *correct protocol converter* is given with the requirement that it must satisfy all the conformity, liveness and transparency properties. It is believed that the converter is useful only when these three properties are all met. The next two sections (Sections 3 and Sections 4) demonstrate how this goal can be achieved.

3 Converter Construction

In this section a simple example is used to demonstrate how to apply the proposed five-step algorithm to solve the protocol conversion problem defined in Section 2. For ease in understanding, each step is briefly described and then applied to the example to show the result. Furthermore, as the algorithm is performed step by step, the reasons and ideas behind each step are also discussed. For easy reference, an outline of the algorithm is provided in Appendix.

Before getting into the details of the algorithm, the given protocols, P and Q, and the conversion service requirement, R, are described below They are the given conditions of the protocol conversion problem defined in Definition 3 of Section 2.

The protocol P, as shown in Fig. 4, is an ABP (Alternating Bit Protocol), in which, P_a sends data D0 and D1 alternately to P_b. For each data sent by P_a, an acknowledgement, A0 for D0 and A1 for D1 respectively, is sent by P_b. The protocol user at P_a executes IN to request sending of data and when data arrives at the other end, P_b executes OUT to pass the data to the user at P_b. The ABP is used in a noisy channel: when data is garbled, it will be retransmitted. Note that "+" denotes the reception of a message or acknowledgement and "-" denotes the sending of a message or acknowledgement. The sets of service and peer transitions in P are as follows:

$$\Sigma_{P_a} = \{ \text{IN} \}$$
$$\Sigma_{P_b} = \{ \text{OUT} \}$$
$$\lambda_{P_a} = \{ \text{-D0, -D1, +A0, +A1} \}$$
$$\lambda_{P_b} = \{ \text{+D0, +D1, -A0, -A1} \}$$

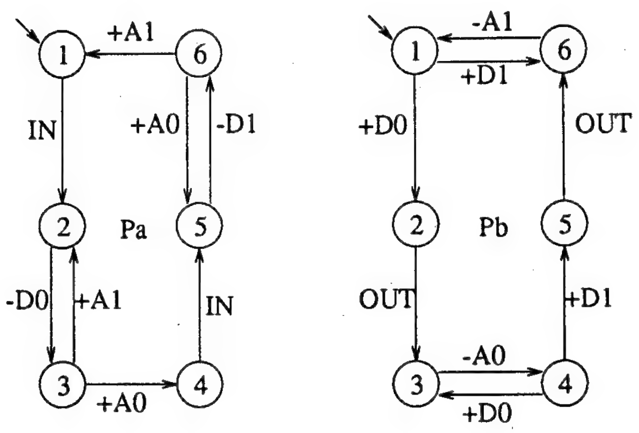


Figure 4: Example 1: The Given Protocol P

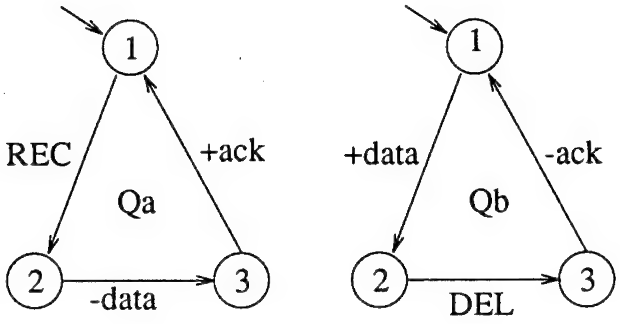


Figure 5: Example 1: The Given Protocol Q

On the other hand, the protocol Q, as shown in Fig. 5, is a non-sequence protocol. Note that this protocol is used in a reliable communication channel, where garbling of data is not possible. As shown in Fig. 5, for each data sent out by Q_a, an acknowledgement, ack, is sent by Q_b. The user at Q_a initiates the service transition REC to hand the data over to Q_a, and when the data arrives at Q_b, it delivers the data to the user at Q_b by executing DEL. The sets of service and peer transitions in Q are as follows:

$$\Sigma_{Q_a} = \{\text{ REC }\}$$
$$\Sigma_{Q_b} = \{\text{ DEL }\}$$
$$\lambda_{Q_a} = \{\text{ -data, +ack }\}$$
$$\lambda_{Q_b} = \{\text{ +data, -ack }\}$$

The service specifications, P_s and Q_s, and the conversion service requirement R are as shown in Fig. 6. The goal in this example is to derive a converter which delivers messages from P_a to Q_b. For ease in discussion, only one way of the traffic (from P_a to Q_b) is considered in this paper; the traffic in the other direction (from Q_a to P_b) can be carried out in the same way.

Note that in this example P is used in a noisy channel and Q in a reliable channel, and the *information* on the channel characteristic is not contained in the service specifications P_s and Q_s. As a result, as mentioned in Section 1 of this paper, P and Q have the same service specification but different implementations. Since the goal of protocol conversion is to derive a converter for the existing implementations of P_s (ABP) and Q_s (non-sequence protocol), one will not be able to construct a *correct* converter for P and Q if no consideration is made on how P_s and Q_s are implemented. This is why it was claimed earlier that the implementation information must be included in the

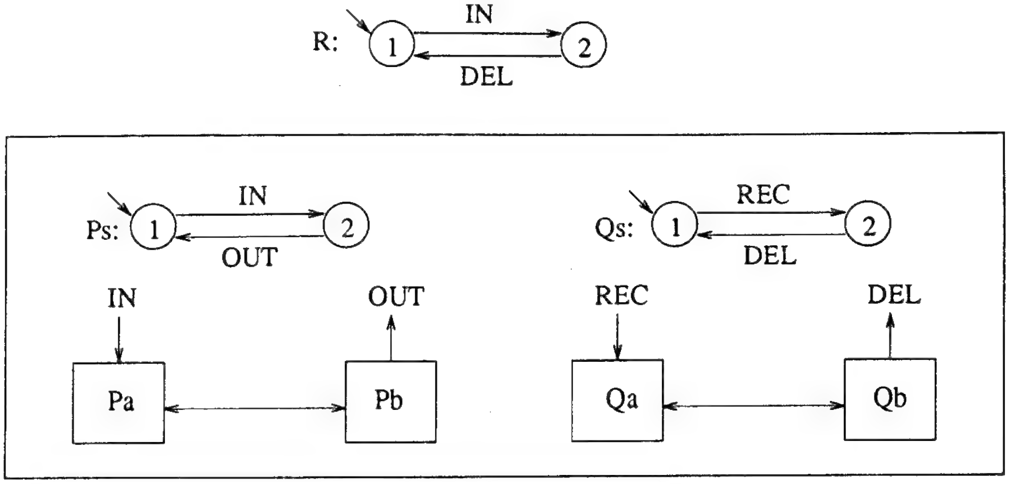


Figure 6: Example 1: The Conversion Service Requirement and Service Specifications

derivation of the protocol converter. (Note that the implementation information is needed at a later stage of the construction process, but not at the first stage of the service specification approach)

3.1 Step 1: Generate the Service System Graph G

In this step the service system graph, G, is generated from the given P_s, R and Q_s by a modified composite operation, $\otimes$.

$$G = < P_s, R, Q_s > = P_s \otimes R \otimes Q_s$$

The operation $\otimes$ is formally defined as follows..

Definition 4. Modified composition operation, $\otimes$, on three CFSM entities. $G = < P_s, R, Q_s > = P_s \otimes R \otimes Q_s$ is a six-tuple $(q, \Sigma, \lambda, \delta, i, f)$, where

1. q is the set of states in G and

   q = {[u, v, w]| u, v, w are states of P_s, R and Q_s respectively }

2. Σ is the finite set of service transitions in G and

   $$\Sigma = \Sigma_{P_s} \cup \Sigma_{Q_s}$$

3. λ is the finite set of peer transitions in G and $\lambda = \Phi$ (λ is empty)

4. δ is the transition function of G and δ([u, v, w], t) =

   (a) [u', v, w] if t $\in \Sigma_{P_s}$ & t $\notin \Sigma_R$ & δ_{P_s}(u, t) = u'

   (b) [u, v, w'] if t $\in \Sigma_{Q_s}$ & t $\notin \Sigma_R$ & δ_{Q_s}(w, t) = w'

   (c) [u', v', w] if t $\in \Sigma_{P_s}$ & t $\in \Sigma_R$ & δ_{P_s}(u, t) = u' & δ_R(v, t) = v'

   (d) [u, v', w'] if t $\in \Sigma_{Q_s}$ & t $\in \Sigma_R$ & δ_R(v, t) = v' & δ_{Q_s}(w, t) = w'

   (e) t is not executable at state [u, v, w] in G if none of the above four conditions is true.

5. i is the initial state of G and i = $[i_{P_s}, i_R, i_{Q_s}]$, where i_{P_s}, i_R and i_{Q_s} are the initial states of P_s, R and Q_s, respectively.

6. f is the set of the final states in G and $f = \{[f_{P_s}, f_R, f_{Q_s}]\}$ $(= \{[i_{P_s}, i_R, i_{Q_s}]\})$

The service system graph G, constructed by the modified composition operation, $\otimes$, describes how the protocols P and Q work together as a composed protocol system to perform the functions requested by the conversion service requirement R. Note that G does not contain any peer transition and it describes the behavior of the composed protocol system at the service transition level (i.e., at SAP). The result of applying the modified composition operation $\otimes$ to P_s, R and Q_s in this example to obtain the service system graph G is shown in Fig. 7.

The difference between the modified composition operation $\otimes$ and the usual composition operation de-

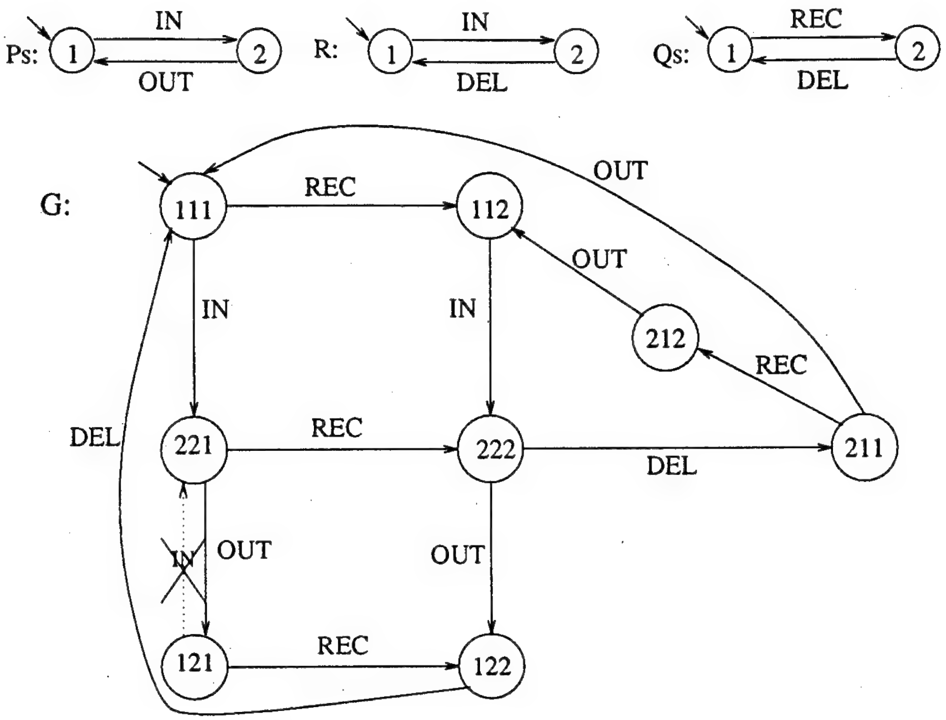


Figure 7: The Service System Graph G of Example 1

fined in [3] is as follows: in the modified composition operation ⊗, a transition t belonging to both P_s and R is executable in G only when both P_s and R are ready to execute it; i.e. both P_s and R have to reach the starting state of t for t to be executable in G. The same is true for those transitions belonging to both R and Q_s. Note that this difference is enforced by items 4.c and 4.d of Definition 4.

As shown in Fig. 7, at state [1, 2, 1], transition IN to state [2, 2, 1] is not executable in G when using the modified composition operation ⊗, since IN $\in \Sigma_{P_s}$ & IN $\in \Sigma_R$ and IN is not executable at state 2 of R, even though IN is executable at state 1 of P_s. On the other hand, in the usual composition operation defined in [3], the transition IN from state [1, 2, 1] to state [2, 2, 1] would have been allowed. The transition sequence

$$p_x = \text{IN.OUT.IN.OUT.REC.DEL}$$

would become a legal path in G, if the transition IN from state [1, 2, 1] to state [2, 2, 1] were allowed in G. It is easy to verify that the transition sequence p_x does not obey the *rule* specified in the conversion service requirement, R, which essentially says that each IN must be followed by *one* DEL in a legal path. To be more specific, the projection of p_x onto Σ_R, $p_x \mid_{\Sigma_R}$ (= IN.IN.DEL), which has two INs and only one DEL, is not accepted by R, and as a result p_x should not be a legal path of G. Therefore, items 4.c and 4.d of Definition 4 are used to exclude from G the transition IN from state [1, 2, 1] to state [2, 2, 1], thereby making the transition sequence p_x (which does not meet the requirement R) an illegal path in G. Note that a legal path of a CFSM in this paper is defined as a sequence of transitions which starts and ends at the initial state.

One may argue that G could lose some *functionalities* from the original protocols if the transition IN from state [1, 2, 1] to state [2, 2, 1] were not included. The answer to this argument is that the *capability* of executing the transition IN is not taken out from G (hence no loss of functionality) after applying the modified composition operation ⊗ to P_s, R and Q_s. The transition IN is indeed included at some other starting states (state [1, 1, 1] and state [1, 1, 2] but not state [1, 2, 1]) to make the final service system graph G obey all the rules specified by P_s, R and Q_s. By going through G, it is clear that the legal paths accepted by G are a mixture of the transitions in P_s

and Q_s in an order that follows the sequencing prescribed by the conversion service requirement R and the given service specifications P_s and Q_s. In short, G describes how the protocols work together to provide the service required, under the constraints of the given conversion service requirement R.

3.2 Step 2: Derive the Conversion Service Specification M

As mentioned earlier, the conversion service requirement R (hence also the derived service system graph G) only describes how the final protocol system should function and neither R nor G tells specifically how the conversion should be performed. The information on how to perform the conversion is, however, implied in G, and the goal of this step is to derive this information from G. Note that the conversion can happen only between P_b and Q_a. To know how the conversion can be done, the relationship between the transitions in P_b and Q_a must be derived first. By projecting G onto $\Sigma_{P_b} \cup \Sigma_{Q_a}$ one can derive a conversion service specification M, which has only transitions of P_b and Q_a and describes the necessary sequencing of the transitions in P_b and Q_a to provide the service required in R; i.e. $M\ (= G\ |_{\Sigma_{P_b} \cup \Sigma_{Q_a}})$ describes how the service transitions of P_b and Q_a *interact* to perform the service requested in R. The projection operation, |, which is the same as that defined in [3], is formally defined as follows.

Definition 5. Project Operation, |.
$M = G\ |_{\Sigma_{P_b} \cup \Sigma_{Q_a}}$ is a CFSM derived from G by the following steps:

1. Change the transitions of G not belonging to $\Sigma_{P_b} \cup \Sigma_{Q_a}$ into null transitions.
2. Removing all null transitions using the algorithm described in [22].

As a matter of fact, in this step the adaptor conversion model is being adopted. The adaptor conversion model is shown in Fig. 8, where the adaptor performs the conversion by synchronizing the execution of the service transitions of P_b and Q_a; i.e. the conversion service specification M specifies the proper (service) transition sequence of the adaptor for the protocol system to perform the functions requested by the requirement R. Note that M (hence also the adaptor) contains only service transitions and all service transitions are later to be removed from the final converter. In Step 5 of the algorithm when all the service transitions are removed from the converter, Fig. 8 is eventually transformed into the converter conversion model depicted in Fig. 9.

The result of applying the projection operation to the service system graph G obtained in Step 1 to derive the conversion service specification M is shown in Fig. 10 and Fig. 11. Part A of Fig. 10 shows the initial projection of G onto $\Sigma_{P_b} \cup \Sigma_{Q_a}$. The removal of states involving null transitions is shown in parts B, C and D of Fig. 10 and Fig. 11. The resulting Fig. 11 shows the Conversion Service Specification, M, obtained after the completion of the project operation.

Note that a state with only one null outgoing transition can be removed simply by changing the ending states of all the incoming transitions into the ending state of the null outgoing transition as shown in part A to part B of Fig. 10, where state 4 is removed. Similarly, a state with only one null incoming transition can be removed simply by changing the starting states of all the outgoing transitions into the starting state of the null incoming transition as shown in part B to part C and part C to part D of Fig. 10, where state 7 is removed in part C and state 2 is removed in part D. Examining Fig. 11, it is obvious that the transition sequence REC.OUT accepted by M is semantically incorrect, since it means that Q_a can send out the data to Q_b before P_b hands over the data received from P_a to Q_a. However, by examining the given service specifications P_s and Q_s and the conversion service requirement R, what they have specified are as follows:

P_s: an IN must be followed by an OUT (IN → OUT).

R: an IN must be followed by a DEL (IN → DEL).

Q_s: a REC must be followed by a DEL (REC → DEL).

Note that M is a projection of G and REC.OUT is just a sub-sequence of transitions of some legal paths of G. The transition sequence REC.OUT does not violate the rules as long as an IN was executed earlier and a DEL is executed later in the original legal paths of G (e.g. IN.REC.OUT.DEL). The given specifications do not specify specifically what the correct ordering of OUT and REC should be, and the ordering of OUT and REC is left to be determined by the implementation of the converter from the service users' viewpoint. Therefore, this further shows the validity of the claim made earlier that the information from service specification alone is not sufficient to derive the

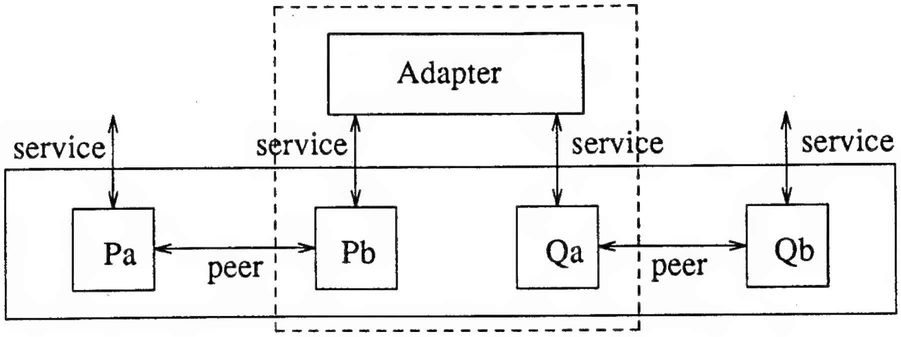


Figure 8: The Adaptor Conversion Model

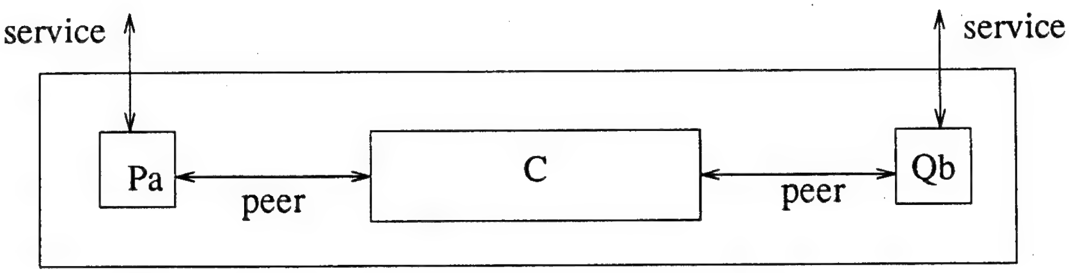


Figure 9: The Converter Conversion Model

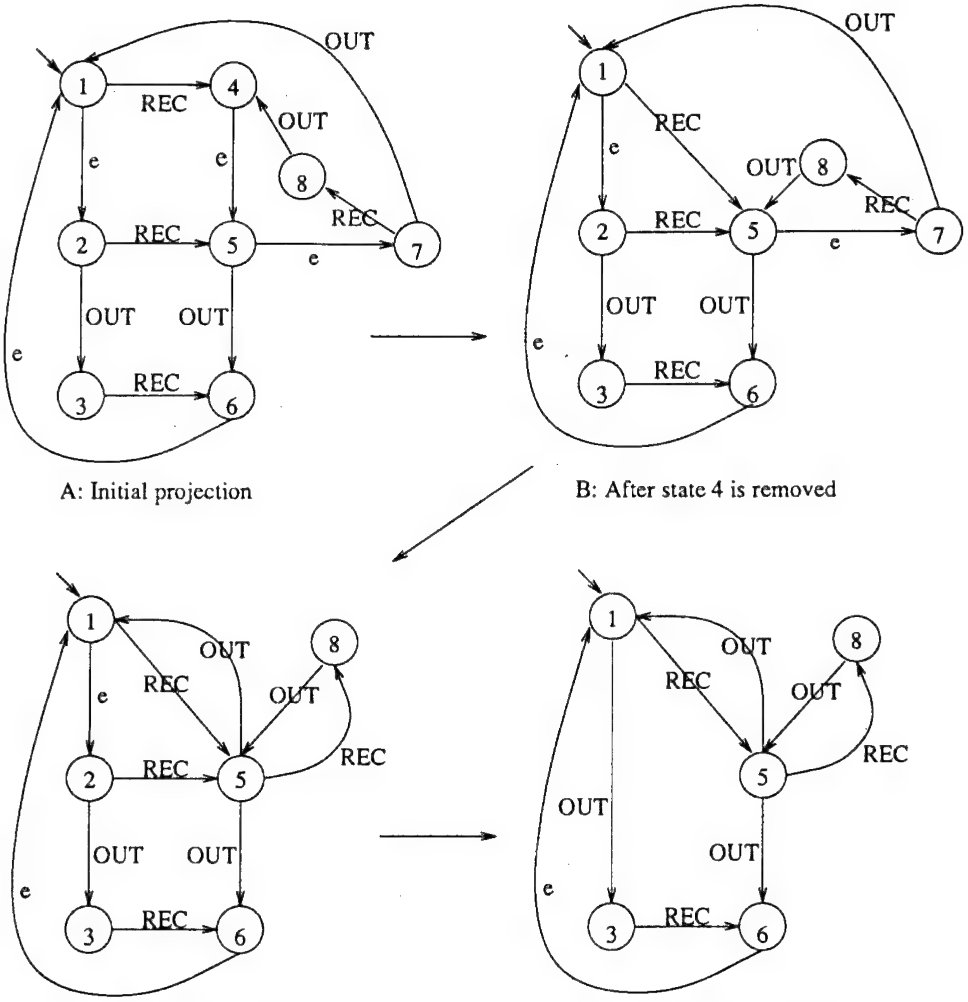
correct protocol converter and the information from the implementation must be included. Nevertheless, M tells us that the transitions OUT and REC need to be synchronized and this is what the next step of the algorithm is for.

3.3 Step 3: Generate the Synchronizing Transition Sets

The goal of this step is to derive the synchronizing transition sets from the conversion service specification M obtained in Step 2. A synchronizing transition set contains the transitions which must be executed *together* (in both P_b and Q_a) for the protocol system to correctly perform the conversion as requested by the service requirement R. For ease in discussion, the transitions in a synchronizing transition set is called the *synchronizing transitions*. Essentially, the idea befind this step is to identify the aforementioned synchronizing transitions and make them executed together such that the other transitions before and after them in both protocol entities P_b and Q_a can be executed synchronously. Fig. 12, in which t_j and t_k are members of a synchronizing transition set, demonstrates how the synchronizing transitions t_j and t_k synchronize the execution of the transitions in protocol entities P_b and Q_a.

As shown in Fig. 12, when P_b and Q_a are forced to execute t_j and t_k at the same time, all the transitions t_m's that are executable only after t_j in P_b should not be executed before t_k in Q_a is executed. For the same reason, the execution of the transitions t_n's in Q_a (which are executable only after t_k) is synchronized with that of t_j in P_b. Consequently, the execution of the transitions t_u's (t_v's) is synchronized with that of t_n's (t_m's) which belong to a different entity. Note that t_m, t_n, t_u and t_v are as shown in Fig. 12.

The complete formal procedure to generate the synchronizing transition sets is given in Appendix of the paper. In this sub-section, the essential steps for the generation of the synchronizing transition sets from M is briefly described: first, all the legal paths p_i's of M are generated, where a legal path of M is a sequence of service transitions which starts from the initial state of M and ends also at the initial state; if any two paths, p_i and p_j, are two different permutations of



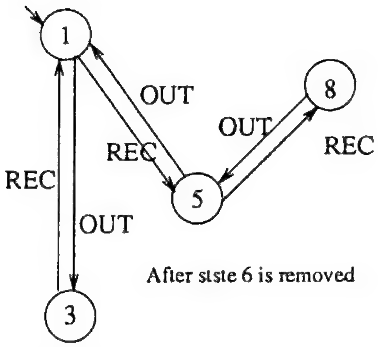
A: Initial projection

B: After state 4 is removed

C: After state 7 is removed

D: After state 2 is removed

Figure 10: The Projection Operation



After stste 6 is removed

Figure 11: The Conversion Service Specification : M

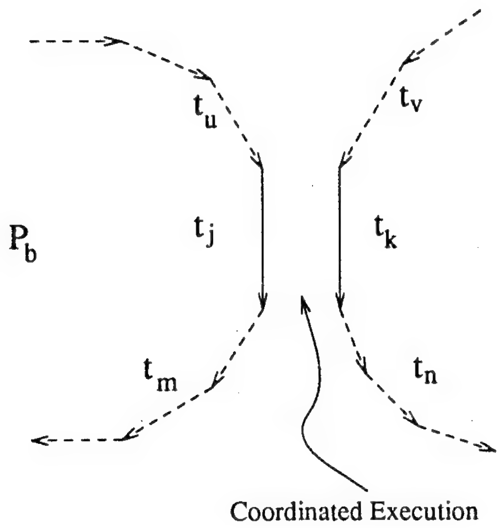


Figure 12: Concurrent Execution with Coordination of t_j and t_k

each other (i.e. different ordering of the same transitions), a synchronizing transition set S_k containing all the transitions of p_i (p_j) is then created; repeat the process until all possible p_i and S_k's are found (i.e. each possible ordering of the transitions in Σ_M is covered by at least one p_i). Note that for two paths p_i and p_j to be different permutations of each other, they must have the same length and contain exactly the *same* transitions; e.g. if p_i contains 4 t_1's and 3 t_2's, p_j must contain the same numbers of t_1's and t_2's, where t_1 and t_2 are different transitions.

The application of this procedure to the example is given as follows. Note that the aforementioned synchronizing transition set is formally defined in Appendix. The legal paths of M are generated as follows:

1. p_1 = (OUT.REC)*
2. p_2 = (REC.OUT)*
3. p_3 = REC.(REC.OUT)*.OUT

The transition sets for each path respectively are as follows:

1. T_1 = { REC, OUT }
2. T_2 = { REC, OUT }
3. T_3 = { REC, OUT }

Following the definition of a synchronizing transition set given in Appendix, it is clear that T_1 (T_2) is a synchronizing transition set since T_1 and T_2 are equal sets, and p_1 and p_2 are different permutations of each other. Thus, the synchronizing transition set S_1 can be identified as follows:

{ REC, OUT }

Since p_1 and p_2 cover all the possible ordering (either an OUT before a REC or the reverse) of all the transitions in Σ_M (REC and OUT), there is no need to proceed further; i.e. there is no need to check on T_3. Note that in this example, T_3 happens to be the same synchronizing transition set as well. In addition, it is not true that every transition set found in this step is always a synchronizing transition set. The transition set T_1 obtained in the example in Section 4 is not a synchronizing transition set.

One may argue that it is so trivial that REC and OUT should be synchronized and this step of the algorithm is redundant. This may ne true for this particular example, in which there is only one service transition in each of P_b and Q_a (hence only 2 different transitions in M); therefore, there is at most one synchronizing transition set, and these 2 transitions are both in the synchronizing transition set. However, in situations in which there are more service transitions in M (for complex protocols), finding the synchronizing transition set is not always trivial, and a formal procedure to generate the synchronizing transition sets is needed, as given in Appendix. An example to illustrate this situation is shown in Section 4.

3.4 Step 4: Compose the Converter Using the Coordinated Composition Operation

In this step the coordinated composition operation, $\odot$, is used to derive the converter, using the synchronizing transition sets obtained in Step 3 as the synchronizing mechanism. This is also the step in which information from the protocol implementation is involved in the construction of the converter. The idea of this step is to let protocol entities P_b and Q_a execute concurrently and coordinatedly. As depicted in Fig. 12, when the coordinated composition operation, $\odot$, is applied to the protocol specifications of P_b and Q_a, protocol entities P_b and Q_a are forced to execute the transitions in synchronizing transition sets together and the coordination/synchronization between transitions in P_b and Q_a is achieved.

In this example, there is only one synchronizing transition set S_1. For ease in discussion, a new transition V_1 for S_1 is created, such that the execution of V_1 in both P_b and Q_a means the execution of all the transitions in S_1 (i.e. REC and OUT) at the same time. That is, V_1 is to be incorporated into the converter by the $\odot$ operation. The new transition V_1 is called the *virtual atomic transition*, and the following can be established:

virtual atomic transition V_1 with
$S_1 = \{$ REC, OUT $\}$
and
converter $C = P_b \odot Q_a$

The operation $\odot$ is defined as follows.

Definition 6. Coordinated Composition Operation $\odot$.
The converter $C = P_b \odot Q_a$, with the virtual atomic transition V_i and the corresponding synchronizing transition set S_i obtained in Step 3, where i = 1, 2, ..., is a six-tuple $(q, \Sigma, \lambda, \delta, i, f)$, where

1. q is the set of states in C and

   $q = \{[u, v] |$ u, v are states of P_b and Q_a respectively $\}$

2. Σ is the finite set of service transitions in C and

   $$\Sigma = \Sigma_{P_b} \cup \Sigma_{Q_a} \cup \{ \text{ all } V_i\text{'s } \}$$

   (The service transition set in C will become empty when all the service and virtual atomic transitions are removed from the final converter at the last step of the algorithm in Section 3.5)

3. λ is the finite set of peer transitions and

   $$\lambda = \lambda_{P_b} \cup \lambda_{Q_a}$$

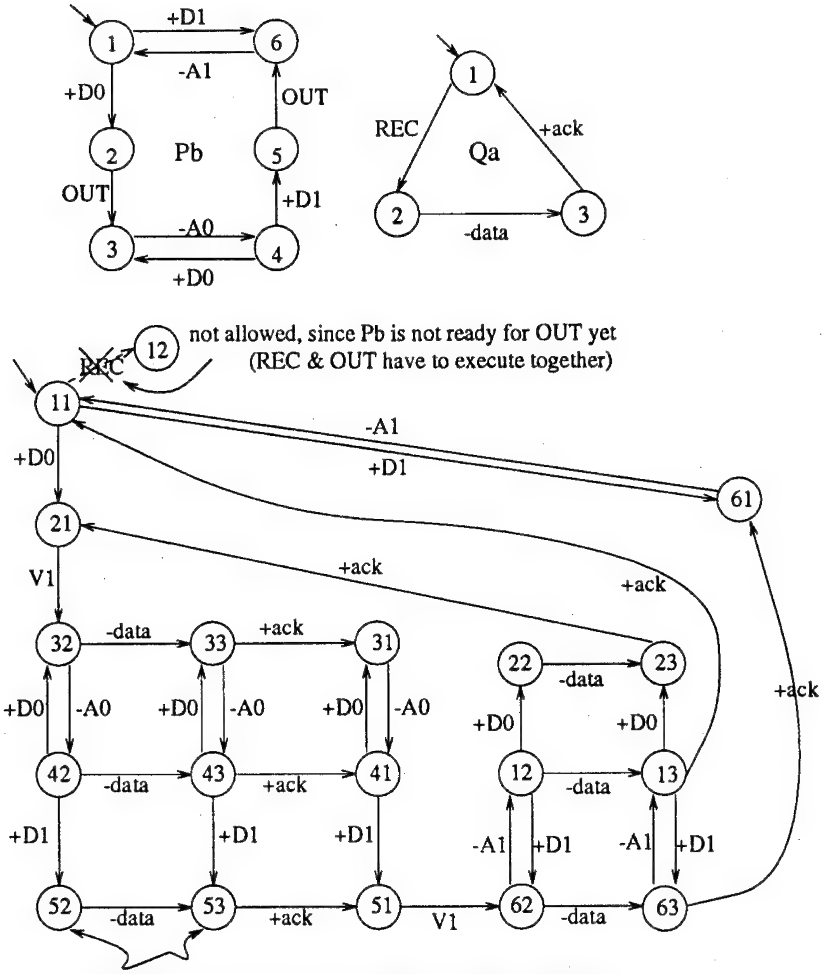
4. δ is the transition function of C and δ([u, v], t) =
   (a) [u', v] if t $\in \lambda_{P_b}$ & δ_{P_b}(u, t) = u'
   (b) [u, v'] if t $\in \lambda_{Q_a}$ & δ_{Q_a}(v, t) = v'
   (c) [u', v] if t $\in \Sigma_{P_b}$ & t $\notin$ any of the S_i's & δ_{P_b}(u, t) = u'
   (d) [u, v'] if t $\in \Sigma_{Q_a}$ & t $\notin$ any of the S_i's & δ_{Q_a}(v, t) = v'
   (e) [u', v'] if t is a virtual atomic transition, V_i,
   & δ_{P_b}(u, t') = u', where t' $\in \Sigma_{P_b}$
   and t' $\in S_i$
   & δ_{Q_a}(v, t") = v', where t" $\in \Sigma_{Q_a}$
   and t" $\in S_i$.
   (In this example, t' = OUT and t" = REC)
   (f) t is not executable at state [u, v] in C if none of the above is true.

5. i is the initial state of C and $i = [i_{P_b}, i_{Q_a}]$, where i_{P_b} and i_{Q_a} are the initial states of P_b and Q_a, respectively.

6. f is the set of the final states in C and $f = \{[f_{P_b}, f_{Q_a}]\} \ (= \{[i_{P_b}, i_{Q_a}]\})$

The result of applying the $\odot$ operation to P_b and Q_a to obtain the Converter C is shown in Fig. 13.

The $\odot$ operation allows all the transitions in P_b and Q_a to execute concurrently in the sequence as the original protocols prescribed, as long as the transitions are not synchronizing transitions (i.e. transitions which do not belong to a synchronizing transition set). The synchronizing transitions in one entity have their chance of execution when the peer entity is ready to execute the corresponding synchronizing transitions in the same synchronizing transition set. In other words, one entity must wait for its peer to execute together the synchronizing transitions in a synchronizing transition set. Note that a synchronizing transition set always contains transitions from both peer entities. As shown in Fig. 13, at state [1, 1], Q_a must wait for P_b to get to state 2 before executing REC (V_1); i.e. at state [2, 1] of C, P_b (in state 2) is ready to execute OUT and Q_a (in state 1) is ready to execute



not allowed, since Pb is not ready for OUT yet (REC & OUT have to execute together)

OUT can not be executed since Qa is not ready for REC yet

Figure 13: The Converter C

REC, and both OUT and REC belong to the synchronizing transition set S_1. The waiting, however, does not change the ordering of the transition executions as prescribed in the original P_b and Q_a. In addition, none of the transitions is thrown away by the $\odot$ operation. As a result, the properties and functionalities of the original protocols are preserved.

3.5 Step 5: Remove the remaining service transitions

This is the final step to derive the protocol converter. The converter obtained in Step 4 contains both service and virtual atomic transitions. Recall that there are no service users on top of the converter; therefore, all the service transitions must be changed to null transitions and then removed from the final converter using the algorithm described in [22]. Moreover, in this example, the virtual atomic transition V_1 must also be removed since the execution of V_1 means the execution of service transitions only, i.e. OUT and REC. The resulting final converter specification is shown in Fig. 14. Note that the same technique described in Section 3.2 can be used to remove the null transitions (which are service transitions and virtual atomic transitions in this example).

To conclude this example, let us see what has been achieved during the construction. Notice that the behavior of either P_b or Q_a has not been changed. In-

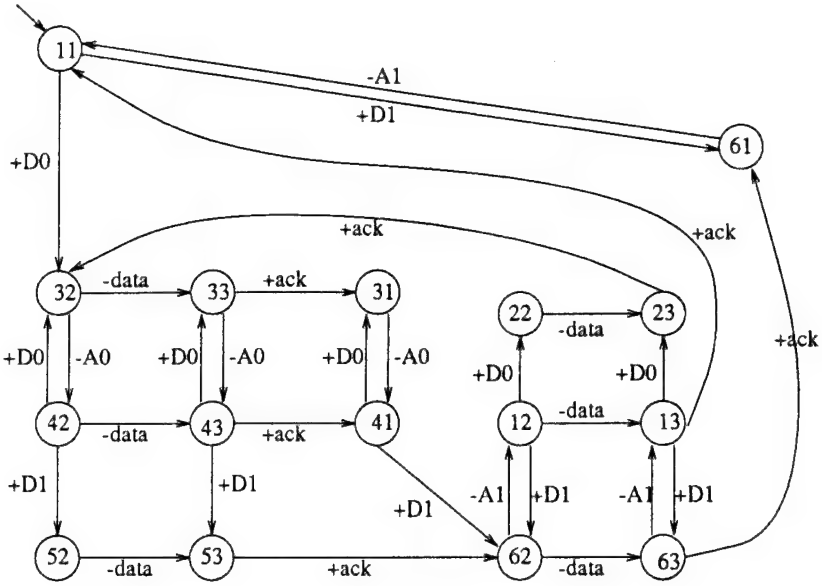


Figure 14: The Final Converter of Example 1

stead, the execution order of the transitions between P_b and Q_a has been forced in accordance with the conversion service requirement, R (hence also M). All transition sequences accepted by P_b (Q_a) are also accepted by the converter and vice versa (in terms of the projection of the legal paths of the final converter specification) while the service transitions are considered null transitions in the converter. Therefore, the properties and functionalities of the original protocols are preserved and the three properties for correctness are satisfied with no need for an additional validation phase.

4 A More Complex Example

In this section, a more complex example is presented to demonstrate that finding the synchronizing transition sets is not always trivial and a formal procedure in Step 3 of the proposed algorithm is needed. Also presented in this section are some strategies that can be used to reduce the complexity in Step 3 and Step 4 of the algorithm. Moreover, the synchronizing transition set in this example has more than 2 transitions; therefore, the virtual atomic transition must be expanded into a sequence of peer transitions at the last step (Step 5) of the converter construction process instead of just removing it.

Fig. 15 shows the given protocol specifications of P and Q, in which the service transitions are labeled in capital letters. Protocol P is a simplified flow control protocol. Before sending each data from P_a to P_b, a reservation needs to be confirmed. If P_b has no space available to receive data, it can send a choke message to P_a to stop the sending. Protocol Q is a simple non-sequence protocol as in the previous example. The service specifications, P_s, Q_s, and the conversion requirement R are as shown in Fig. 16.

The result of applying the $\otimes$ and $|$ operations to P_s, R and Q_s to obtain the conversion service specification M at Step 2 of the algorithm is shown in Fig. 17.

Following the procedure outlined in Appendix for Step 3, a legal path can be obtained as follows:

$$p_1 = \text{REQ.FULL}$$
(and the transition set $T_1 = \{\text{REQ, FULL}\}$)

where p_1 contains transitions from P_b only, which means that the transitions (REQ and FULL) in p_1 have nothing to do with the synchronization between protocols P and Q; thus, REQ and FULL can be removed from M. The simplified M is shown in Fig. 18, where states [2, 2, 1] and [3, 2, 2] are removed. Note that, as M becomes smaller, so does the complexity of this step (Step 3) in the generation of synchronizing transition sets. For the same reason and from the fact that there are no service users on top of the converter, REQ and FULL can also be removed from the original

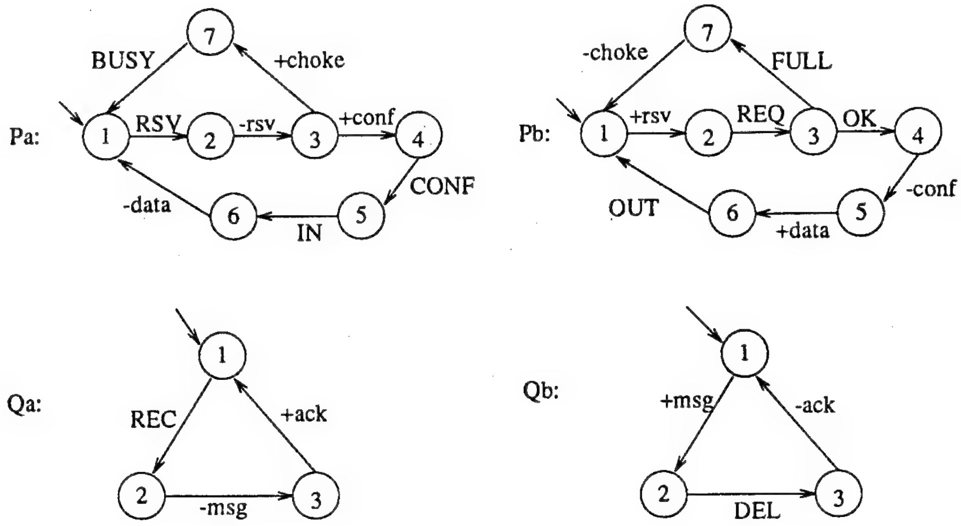


Figure 15: The Given Protocols P and Q of Example 2

protocol specification P_b in Fig. 15 (before composition of the final converter at Step 4). The simplified P_b is shown in Fig. 19, where REQ and FULL are removed from P_b and only service transitions OK and OUT remain in P_b (as the synchronizing mechanism) for the next step (Step 4) of the algorithm. With the simplified protocol specification of P_b as an operand of the coordinated composition operation $\odot$, the complexity of Step 4 can also be reduced.

Following through the procedure in Step 3 of Appendix, a synchronizing transition set S_1 = {OK, OUT, REC} is obtained. Note that in S_1, OK and OUT belong to P_b and REC belongs to Q_a. This means one service transition of Q_a, REC, must be synchronized with two service transitions of P_b, OK and OUT. Thus, a virtual atomic transition V_1 for S_1 can be created. The meaning of the execution of V_1 in this example is quite different from that in Example 1, since, by definition, the execution of V_1 (in this example) means execution of more than just one transition from each peer protocol entity (OK and OUT in P_b and REC in Q_a). Following the protocol specification of P_b in Fig. 19, it is easy to see that a sequence of transitions that execute both OK and OUT from P_b is:

$$\rho_{P_b} = \text{OK.-conf.+data.OUT}$$

In other words, to execute both OK and OUT together with REC in Q_a, a sequence of transitions ρ_{P_b} must be executed atomically. To be more specific, the execution of V_1 means the execution of both ρ_{P_b} in P_b and REC in Q_a atomically. Thus, the meaning of the execution of a virtual atomic transition is extended (to cover the execution of a sequence of transitions), and the definition of the coordinated composition operation $\odot$ can be refined accordingly as follows:

Definition 7. Refined Coordinated Composition Operation $\odot$.
The converter $C = P_b \odot Q_a$, with the virtual atomic transition V_i, the corresponding synchronizing transition set S_i and a legal path p_i (from the conversion service specification M) obtained in Step 3, where i = 1, 2, ..., is a six-tuple $(q, \Sigma, \lambda, \delta, i, f)$, where:

1. q is the set of states in C and

   q = {[u, v]| u, v are states of P_b and Q_a, respectively }

2. Σ is the finite set of service transitions in C and

   $$\Sigma = \Sigma_{P_b} \cup \Sigma_{Q_a} \cup \{ \text{ all } V_i\text{'s } \}$$

   (Again, the service transition set Σ will become empty at the last step (Step 5))

3. λ is the finite set of peer transitions and

   $$\lambda = \lambda_{P_b} \cup \lambda_{Q_a}$$

4. δ is the transition function of C and δ([u, v], t) =

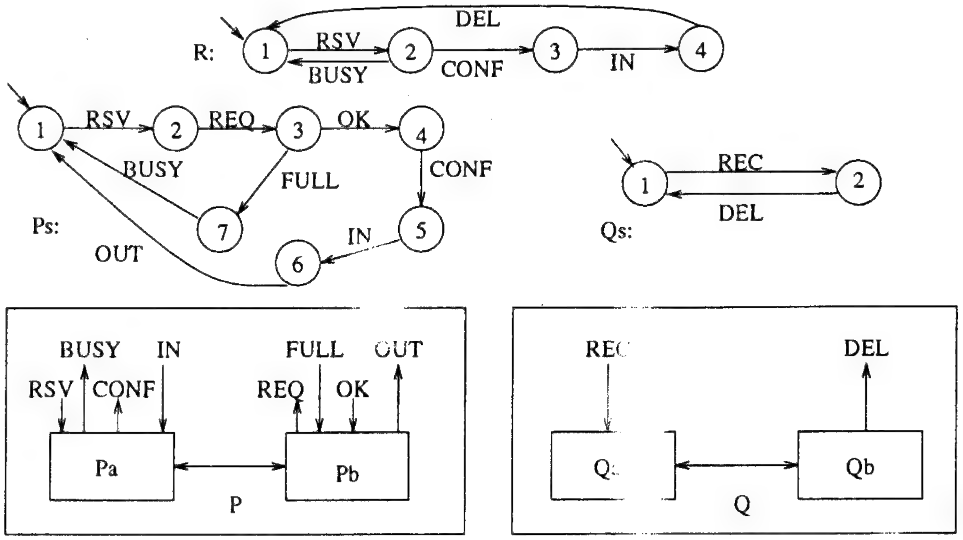


Figure 16: The Conversion Service Requirement and the Service Specifications

(a) [u', v] if t ∈ λ_{P_b} & δ_{P_b}(u, t) = u'

(b) [u, v'] if t ∈ λ_{Q_a} & δ_{Q_a}(v, t) = v'

(c) [u', v] if t ∈ Σ_{P_b} & t ∉ any of the S_i's & δ_{P_b}(u, t) = u'

(d) [u, v'] if t ∈ Σ_{Q_a} & t ∉ any of the S_i's & δ_{Q_a}(v, t) = v'

(e) [u', v'] if t is a virtual atomic transition, V_i,
& $\delta^*_{P_b}$(u, ρ_{P_b}) = u', where $\delta^*_{P_b}$ means applying δ_{P_b} to a sequence of transitions and ρ_{P_b} is a legal sequence of transitions in P_b, which starts and ends with service transitions that are members of S_i and $\rho_{P_b} |_{\Sigma_{P_b}} = p_i |_{\Sigma_{P_b}}$
& $\delta^*_{Q_a}$(v, ρ_{Q_a}) = v', where $\delta^*_{Q_a}$ means applying δ_{Q_a} to a sequence of transitions and ρ_{Q_a} is a legal sequence of transitions in Q_a, which starts and ends with service transitions that are members of S_i and $\rho_{Q_a} |_{\Sigma_{Q_a}} = p_i |_{\Sigma_{Q_a}}$
(ρ_{P_b} = OK.-conf.+data.OUT and ρ_{Q_a} = REC for V_1 in Example 2)
Note that a legal sequence of transitions of a CFSM is a sub-sequence of a legal path of the CFSM.

(f) t is not executable at state [u, v] in C if none of the above conditions is true.

5. i is the initial state of C and i = [i_{P_b}, i_{Q_a}], where i_{P_b} and i_{Q_a} are the initial states of P_b and Q_a, respectively.

6. f is the set of the final states in C and f = {[f_{P_b}, f_{Q_a}]} (= {[i_{P_b}, i_{Q_a}]}).

The result of applying the refined coordinated composition operation to protocol entities P_b and Q_a to obtain the converter is shown in Fig. 20. Note that the simplified protocol specification of P_b in Fig. 19 is used in this step to reduce the complexity.

In this example, a sequence of transitions from one protocol entity is converted into a transition in another. In other words, using the refined coordinated composition operation defined above, it is possible to perform the protocol conversion between sequences of transitions in different protocols. This is a very encouraging result of the proposed algorithm since many of the existing conversion algorithms can only handle mapping between single transition/message (i.e. one-to-one conversion in terms of either transitions or messages), and the semantics of a sequence of transitions/messages can be different from that of a simple concatenation of the semantics of each original individual transition/message for complex protocols. It is our observation that a conversion algorithm capable of handling conversion between sequences of transitions is more powerful and practical when dealing

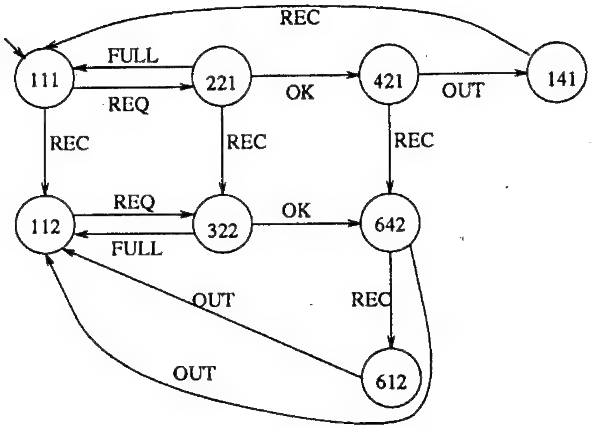


Figure 17: The Conversion Service Specification M of Example 2

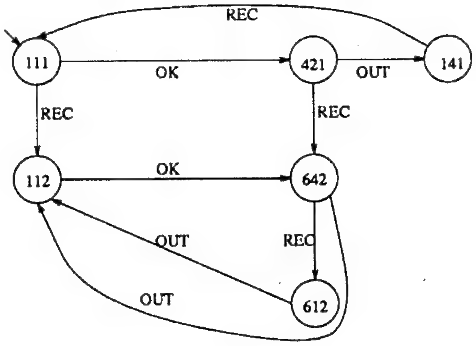


Figure 18: The Simplified Conversion Service Specification M of Example 2

with complex protocols. Moreover, note that the new transition V_1 has become a ***virtual atomic transition*** in the sense that when V_1 is executed all transitions in both ρ_{P_b} (a sequence of transitions) and ρ_{Q_a} are executed atomically. Also note that V_1's execution means not only execution of service transitions but also some peer transitions (-conf.+data). Therefore, V_1 must be expanded to incorporate the peer transitions from ρ_{P_b} in the final converter specification. The final converter is shown in Fig. 21. Note that a dummy state, D, is created for the expansion of the virtual atomic transition V_1.

5 Formal Proof of Correctness

In this section a formal proof is presented to show that the proposed algorithm always constructs a converter that satisfys the three properties of a *correct* converter as defined in Definition 3 of Section 2.

For the transparency property, it is easy to see that the protocol entities at the end nodes (P_a and Q_b) of the composed protocol system are not altered during the converter construction; therefore, the transparency property is indeed satisfied. In addition, the liveness and conformity properties are satisfied (without the need of a validation phase) since all the properties and functionalities from the original protocol entities are preserved during the derivation of the converter. Intuitively, since all the properties and functionalities are preserved and inherited by the converter, it will have the same properties and perform the same functions as the original protocols do. In other words, it will inherit the liveness property if the original protocols do have the liveness property. Similarly, it will also inherit all the functionalities of

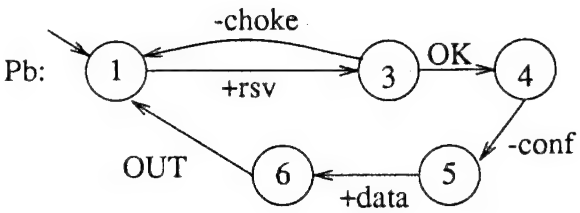


Figure 19: The Simplified Protocol Specification P_b of Example 2

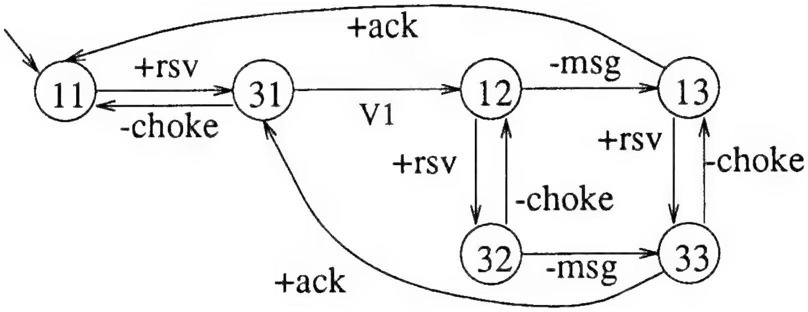


Figure 20: The Converter with Virtual Atomic Transition V_1

the original protocols and hence satisfy the conformity property. Formally, the rest of this section shows a formal proof of the following statement: the liveness and conformity properties are preserved by the converter and an additional validation phase is not required.

5.1 Liveness Property

As mentioned in Section 2, the liveness property of a protocol converter means that the composed protocol system, P_a, C, and Q_b, is deadlock/livelock free. Formally, the liveness property between three protocol entities is defined in Definition 8 as follows.

> **Definition 8.** Liveness property among three protocol entities.
> In a composed protocol system of P_a, C and Q_b, where P_a and Q_b are protocol entities at the end nodes and C is the converter between them, if it is deadlock/livelock free between P_a and C and between C and Q_b, the converter C (also the composed protocol system) is said to have the liveness property.

The goal of this subsection is to prove that the converter constructed by the proposed algorithm does have the liveness property, and the following definition of a *live* converter will be used in the formal proof.

> **Definition 9.** A *live* converter for protocol P and Q.
> A converter C is a *live* converter iff the following is true:

$\forall$ a path γ in C, where
$\delta^*_C([i_{P_b}, i_{Q_a}], \gamma) = [u,\ v]$,
(i.e. γ starts from the initial state $[i_{P_b},\ i_{Q_a}]$
and ends at a state [u, v])
$\exists$ paths τ of P_b and ω of Q_a $\ni$
$\gamma\,|_{\lambda_{P_b}} = \tau\,|_{\lambda_{P_b}}$ & $\delta^*_{P_b}(i_{P_b}, \tau) = u$
and

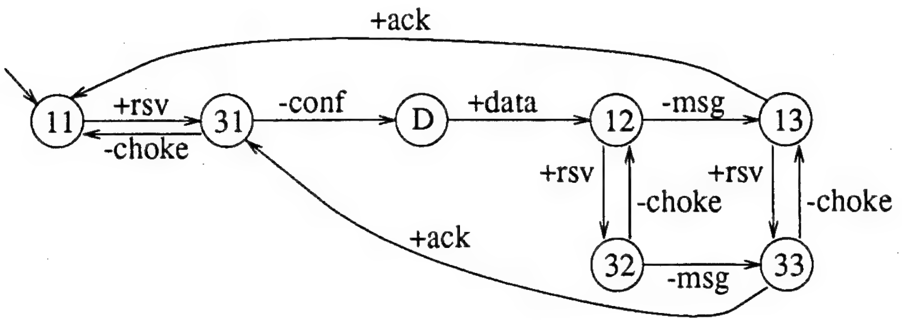


Figure 21: The Final Converter with Virtual Atomic Transition Expanded

$$\gamma \mid_{\lambda_{Q_a}} = \omega \mid_{\lambda_{Q_a}} \ \& \ \delta^*_{Q_a}(i_{Q_a}, \omega) = v$$

Note that a path in this paper is a transition sequence which starts from the initial state of the CFSM and $[i_{P_b}, i_{Q_a}]$ is the initial state of C (i_{P_b} and i_{Q_a} are the initial states of P_b and Q_a, respectively). δ^* means applying δ to a sequence of transitions as defined in Definition 7.

Next, it is shown in Lemma 1 that a live converter has the liveness property if the original protocols P and Q are deadlock/livelock free and in Lemma 2, the converter constructed by the proposed algorithm is proved to be a live converter.

Lemma 1. A live converter C has the liveness property if both P and Q have the liveness property.

Given:
1. P and Q are deadlock and livelock free.
2. C is a live converter.

Need to Prove:
It is contradictory to the given conditions if C does not have the liveness property.

Proof:

$\Longrightarrow$ Assume C does not have the liveness property

$\Longrightarrow$ $\exists$ a path γ of C $\ni$ when γ is executed at C, the entities C, P_a and Q_b reach a deadlock/livelock state among them

$\Longrightarrow$ $\exists$ paths γ of C, α of P_a, β of Q_b $\ni$ when γ, α and β are executed the entities C, P_a and Q_b reach a deadlock/livelock state among them (either between P_a and C or between C and Q_b)

$\Longrightarrow$ (Since C is a live converter) $\exists$ paths τ of P_b and ω of Q_a, where:

$$\gamma \mid_{\lambda_{P_b}} = \tau \mid_{\lambda_{P_b}}$$
$$\&$$
$$\gamma \mid_{\lambda_{Q_a}} = \omega \mid_{\lambda_{Q_a}}$$

$\ni$ when α, τ, ω, β are executed at P_a, P_b, Q_a and Q_b respectively, deadlock/livelock is resulted between either P_a and P_b or Q_a and Q_b

$\Longrightarrow$ Either P or Q is not deadlock/livelock free

$\Longrightarrow$ In contradiction to the given condition 1

In Lemma 2, the induction method on the length of the transitions accepted by protocol entities is used to prove that the converter C' constructed at Step 4 is a live converter. Note that the converter constructed at Step 4 contains service transitions which are eventually removed at Step 5 of the algorithm.

Lemma 2. The converter C' constructed at Step 4 is a live converter; i.e.

$$\forall \text{ a path } \gamma \text{ of } C', \text{ where}$$
$$\delta^*_{C}([i_{P_b}, i_{Q_a}], \gamma) = [u, v],$$
$$\exists \text{ paths } \tau \text{ of } P_b \text{ and } \omega \text{ of } Q_a \ni$$
$$\gamma \mid_{\lambda_{P_b}} = \tau \mid_{\lambda_{P_b}} \ \& \ \delta^*_{P_b}(i_{P_b}, \tau) = u$$
$$\text{and}$$
$$\gamma \mid_{\lambda_{Q_a}} = \omega \mid_{\lambda_{Q_a}} \ \& \ \delta^*_{Q_a}(i_{Q_a}, \omega) = v$$

Base:
It is true for γ of length 0

Induction Hypothesis:
Assume it is true for all γ's of length up to m

Need to Prove:
It is true for γ' of length m+1

Proof:

$\Longrightarrow$ (From the induction hypothesis)

$\forall$ γ of length m, $\exists$ paths τ of P_b and ω of Q_a $\ni$

$$\gamma \mid_{\lambda_{P_b}} = \tau \mid_{\lambda_{P_b}} = t_1.t_2...t_l$$
$$\&$$
$$\gamma \mid_{\lambda_{Q_a}} = \omega \mid_{\lambda_{Q_a}} = s_1.s_2...s_n,$$

where $t_1, t_2, ... ,t_l \in \lambda_{P_b}$, and $s_1, s_2, ... ,s_n \in \lambda_{Q_a}$;

$\delta^*_{C'}([i_{P_b}, i_{Q_a}], \gamma) = [u, v]$ and $\delta^*_{P_b}(i_{P_b}, \tau) = u$ and $\delta^*_{Q_a}(i_{Q_a}, \omega) = v$

(i.e. after execution of γ, C' is at state [u, v]; after τ, P_b is at state u, and after ω, Q_a is at state v)

$\Longrightarrow$ Let $\gamma' = \gamma.t$ be a path of length m+1 and t is executable at state [u, v] in converter C':

By definition (Item 4 of Definition 7 in Section 4), t can be either a virtual atomic transition or a peer/service transition of P_b or Q_a. If t is a virtual atomic transition, it can be expanded into a sequence of *normal* service/peer transitions (i.e. into the ρ defined in Item 4.e of Definition 7) without changing the behavior of C'. Thus, without loss of generality, t only needs to be considered as a normal transition of the original protocol entities P_b/Q_a:

1. If t $\in \lambda_{P_b}$, then, by Item 4.a of Definition 7:
   $\rightarrow \delta_{C'}([u, v], t) = [u', v]$ and $\delta_{P_b}(u, t) = u'$
   $\rightarrow \delta^*_{C'}([i_{P_b}, i_{Q_a}], \gamma.t) = [u', v]$ and $\delta^*_{P_b}(i_{P_b}, \tau.t) = u'$
   $\rightarrow \gamma.t \mid_{\lambda_{P_b}} = (\gamma \mid_{\lambda_{P_b}}).t = t_1.t_2...t_l.t = (\tau \mid_{\lambda_{P_b}}).t = \tau.t \mid_{\lambda_{P_b}}$
   $\rightarrow \gamma.t \mid_{\lambda_{P_b}} = \tau.t \mid_{\lambda_{P_b}}$
   (in this case, $\gamma.t \mid_{\lambda_{Q_a}}$ does not change and it is still equal to $\omega \mid_{\lambda_{Q_a}}$)
2. Similarly, by Item 4.b of Definition 7, if t $\in \lambda_{Q_a}$:
   $\rightarrow \gamma.t \mid_{\lambda_{Q_a}} = \omega.t \mid_{\lambda_{Q_a}}$
   ($\gamma.t \mid_{\lambda_{P_b}}$ is still equal to $\tau \mid_{\lambda_{P_b}}$ in this case)
3. If t $\in \Sigma_{P_b}$ then by Item 4.c of Definition 7:
   $\rightarrow \delta_{C'}([u, v], t) = [u', v]$ and $\delta_{P_b}(u, t) = u'$
   $\rightarrow \delta^*_{C'}([i_{P_b}, i_{Q_a}], \gamma.t) = [u', v]$ and $\delta^*_{P_b}(i_{P_b}, \tau.t) = u'$
   $\rightarrow \gamma.t \mid_{\lambda_{P_b}} = \gamma \mid_{\lambda_{P_b}} = \tau \mid_{\lambda_{P_b}}$
   $\rightarrow \gamma.t \mid_{\lambda_{P_b}} = \tau \mid_{\lambda_{P_b}}$
4. Similarly, by Item 4.d of Definition 7, if t $\in \Sigma_{Q_a}$:
   $\rightarrow \gamma.t \mid_{\lambda_{Q_a}} = \omega \mid_{\lambda_{Q_a}}$

$\Longrightarrow$ For $\gamma' = \gamma.t$ of length m+1, where $\delta^*_C([i_{P_b}, i_{Q_a}], \gamma') = [u'', v'']$,
$\exists$ paths τ' of P_b and ω' of Q_a $\ni$

$$\gamma' \mid_{\lambda_{P_b}} = \tau' \mid_{\lambda_{P_b}} \ \& \ \delta^*_{P_b}(i_{P_b}, \tau') = u''$$
$$\text{and}$$
$$\gamma' \mid_{\lambda_{Q_a}} = \omega' \mid_{\lambda_{Q_a}} \ \& \ \delta^*_{Q_a}(i_{Q_a}, \omega') = v''$$

Depending upon whether t is a peer or service transition of P_b, τ' can be either $\tau.t$ or τ and u'' can be either u' or u. The same is true for ω' and v''.

$\Longrightarrow$ It is true for path γ' of length m+1.

From Lemma 1 and Lemma 2, it is easy to see that the converter C' constructed at Step 4 has the liveness property if the original protocols P and Q have the liveness property. Since there are no service users on top of the converter, the service transitions in C' do not serve any functions. Therefore, removing the service transitions from C' to obtain the final converter C at Step 5 does not change the functionalities and properties of the converter. In other words, the final converter does have the liveness property if both P and Q have the liveness property.

Theorem 1. The converter C constructed by the proposed algorithm has the liveness property if the original protocols P and Q have the liveness property.

Proof:
Trivial from Lemma 1 and Lemma 2 above.

5.2 Conformity Property

For proof of the conformity property in this subsection, the strategy is first to show that the converter constructed by the algorithm supports no more functions than the original protocol entities P_b and Q_a do, and then to prove that it also supports no less functions.

Lemma 3. A live converter C supports no more functions than the original protocol entities P_b and Q_a do.

Proof:

$\Longrightarrow$ Since C is a live converter:

$$\forall \text{ a path } \gamma \text{ in C}, \exists \text{ paths } \tau \text{ of } P_b \text{ and } \omega \text{ of } Q_a \ni$$
$$\gamma |_{\lambda_{P_b}} = \tau |_{\lambda_{P_b}} \ \& \ \gamma |_{\lambda_{Q_a}} = \omega |_{\lambda_{Q_a}}$$

$\Longrightarrow$ The function supported by $\gamma |_{\lambda_{P_b}}$ is supported by τ in P_b and the function supported by $\gamma |_{\lambda_{Q_a}}$ is supported by ω in Q_a

$\Longrightarrow$ The function supported by a legal path of C can be supported by a legal path of P_b and a legal path of Q_a

$\Longrightarrow$ C supports no more functions than P_b and Q_a do

Since it has been proved in Section 5.1 that the converter constructed by the algorithm is a live converter, by Lemma 3 above, it is trivial to see that the converter supports no more functions than the original protocol entities P_b and Q_a do. Thus, the following Lemma has been proved:

Lemma 4. The converter C constructed by the proposed algorithm supports no more functions than the original protocol entities P_b and Q_a do.

Proof:
Trivial from Lemma 2 and Lemma 3.

Next, it is shown that the converter constructed supports no less functions than the original protocol entities P_b and Q_a do. The following definition is used in the proof.

Definition 10. A conforming converter for protocol P and Q:
A converter C is a *conforming* converter iff the following is true:

$$\forall \text{ a legal path } \tau \text{ of } P_b \text{ (and } \omega \text{ of } Q_a), \exists \gamma_1 \text{ (and } \exists \gamma_2) \text{ of C} \ni$$
$$\tau |_{\lambda_{P_b}} = \gamma_1 |_{\lambda_{P_b}}$$
$$(\text{and } \omega |_{\lambda_{Q_a}} = \gamma_2 |_{\lambda_{Q_a}})$$

Note that a legal path is a path which ends at the initial state of a protocol entity.

In Lemma 5 and Lemma 6 below, it is shown that a conforming converter supports no less functions than the original protocol entities do, and that the converter constructed by the proposed algorithm is a conforming converter.

Lemma 5. A conforming converter C supports no less functions than the original protocol entities P_b and Q_a do.

Proof:

$\Longrightarrow$ C is a conforming converter and by Definition 10:

$$\forall \text{ a legal path } \tau \text{ of } P_b \text{ (and } \omega \text{ of } Q_a), \exists \gamma_1 \text{ (and } \exists \gamma_2) \text{ of C} \ni$$
$$\tau |_{\lambda_{P_b}} = \gamma_1 |_{\lambda_{P_b}}$$
$$(\text{and } \omega |_{\lambda_{Q_a}} = \gamma_2 |_{\lambda_{Q_a}})$$

$\Longrightarrow$ The function supported by a legal path τ in P_b (ω in Q_a) is supported by γ_1 (γ_2) in C

$\Longrightarrow$ The function supported by a legal path τ (ω) in P_b (Q_a) can be supported in C by a legal path γ_1 (γ_2)

$\Longrightarrow$ C supports no less functions than P_b (Q_a) does.

Using the same method (induction method) as in the proof of Lemma 2 it can be proved that the converter constructed by the algorithm is indeed a conforming converter.

Lemma 6. The converter C' constructed in Step 4 is a conforming converter.

Proof:
Since the proof is similar to that of Lemma 2 (i.e. the induction method), it is not repeated here.

From Lemma 4, Lemma 5 and Lemma 6, it is easy to see that the converter so constructed supports no more and no less functions than the original protocol entities P_b and Q_a do. Thus, the following Theorem 2 is easily established.

Theorem 2. The converter constructed by the proposed algorithm has the conformity property.

As a result of Theorem 1 in Section 5.1 and Theorem 2 in Section 5.2, it is clear now that the proposed algorithm will always generate a converter with the liveness and conformity properties without an additional validation phase, as claimed earlier.

6 Conclusion

The work presented here shows how to derive a *correct* converter formally from the conversion service requirement and the service specifications given by the service users, by using the operations, $\otimes$, $|$ and $\odot$. Following the ordering of the transition execution sequences described in the original protocol entities, the algorithm uses the synchronizing transition sets and the virtual atomic transitions as the synchronizing mechanism to derive a correct converter. The behavior of the original protocols is inherited by the converter constructed and all the properties and functionalities are preserved. This is the main reason why the proposed 5-step algorithm can satisfy the conformity property of a correct converter without an additional validation phase, which is required by many existing algorithms based on the service specification approach.

The service specification approach is adopted in this paper because it is, in general, easier to find a semantically correct conversion service requirement than to find a correct conversion specification in an ad hoc manner by directly examining the implementation of the given protocols. Therefore, the service specification approach is more practical than the conversion specification approach when dealing with complex protocols.

The proposed algorithm *always* finds a converter that satisfies the given conversion service requirement. If the requirement given is semantically correct, so is the converter constructed without a need for validation phase, since the properties and functionalities of the given conversion service requirement are inherited by the converter. In addition, in Section 4 it is demonstrated that the algorithm has the capability of handling conversion between sequences of transitions in different protocols. This capability is very crucial when dealing with complex protocols in which the semantics of sequences of transitions/messages is different from that of a simple concatenation of the semantics of each original individual transition/message. In other words, in a situation where simple one-to-one transition/message mapping between protocols is not sufficient to derive a correct converter and the conversion can only be done in a unit of sequence of transitions/messages, the proposed algorithm can still derive a correct converter succeesfully.

The complexity of the proposed algorithm is at worst exponential in time and space since the algorithm of removing the null transitions in [22] is exponential. However, the complexity can be reduced by the method described in Section 3.2 (i.e. removal of states with only one incoming or outgoing null transition). Areas of interest for future work include reducing the complexity at Step 3 of the algorithm. That is to find a better way to derive the synchronizing transition sets from the conversion service specification M. In addition, a formal method to derive the semantically correct conversion service requirement from the service specifications is also useful.

References

[1] P. Green, "Protocol Conversion," *IEEE Transactions on Communications*, vol. COM-34, pp. 257–668, Mar. 1986.

[2] M. T. Liu, "Protocol Engineering," in *In Advances in Computers* (M.C. Yovits, ed.), vol. 27, pp. 79–195, New York: Academic Press, July 1989.

[3] M. T. Liu, "Protocol Conversion," in *Proc. International Computer Symposium*, (Hsinchu, Taiwan), pp. 82–92, Dec. 1990.

[4] H. J. Jeng and M. T. Liu, "Protocol Conversion: Synchronizing Transition Set Approach in EFSM Model," Technical Report No. OSU-CISRC-1/93-TR-2, The Ohio State University, Columbus, Ohio, Jan. 1993.

[5] P. Francois and A. Potocki, "Some Methods for Providing OSI Transport in SNA," *IBM Journal of Research & Development*, vol. 27, pp. 452–463, Sept. 1983.

[6] I. Groenback, "Conversion between the TCP and Transport Protocol as a Method of Achieving Interoperability between Data Communication Systems," *IEEE Journal on Selected Areas of Communications*, vol. SAC-4, pp. 288–296, Feb. 1986.

[7] J. M. Rodriguez, "An X.PC/TCP Protocol Translator," in *Proc. IEEE INFOCOM 1988*, pp. 308–313, Mar. 1988.

[8] M. T. Rose and D. E. Cass, "OSI Transport Service on Top of TCP," *Computer Networks and ISDN Systems*, vol. 12, pp. 159–173, 1987.

[9] K. Okumura, "A Formal Protocol Conversion Method," in *Proc. ACM SIGCOMM 1986 Symposium*, (Stowe, Vermont), pp. 30–38, Aug. 1986.

[10] W.-S. C. Y-W. Yao and M. T. Liu, "A Modular Approach to Constructing Protocol Converters," in *Proc. IEEE INFOCOM 1990*, (San Francisco), pp. 572–579, June 1990.

[11] S. S. Lam, "Protocol Conversion–Correctness Problems," in *Proc. ACM SIGCOMM 1986 Symposium*, (Stowe, Vermont), pp. 19–28, Aug. 1986.

[12] K. L. Calvert and S. S. Lam, "An Exercise in Deriving a Protocol Converter," in *Proc. ACM SIGCOMM 1987 Symposium*, (Stowe, Vermont), pp. 151–160, Aug. 1987.

[13] S. S. Lam, "Protocol Conversion," *IEEE Transactions on Software Engineering*, vol. SE-14, pp. 353–362, Mar. 1988.

[14] J. C. Shu and M. T. Liu, "A Synchronization Model for Protocol Conversion," in *Proc. IEEE INFOCOM 1989*, (Ottawa, Canada), pp. 276–284, Apr. 1989.

[15] J. Chang and M. T. Liu, "Using Protocol Validation Technique to Solve Protocol Conversion Problems," in *Proc. 9th IPCCC 1990*, (Scottsdale, Arizona), pp. 539–546, Mar. 1990.

[16] J. Chang and M. T. Liu, "An Approach to Protocol Complementation for Interworking," in *Proc. IEEE ICSI 1990*, (Morristown, New Jersey), pp. 205–211, Apr. 1990.

[17] K. L. Calvert and S. S. Lam, "Deriving a Protocol Converter: A Top-Down Method," in *Proc. ACM SIGCOMM 1989 Symposium*, (Austin), pp. 247–258, Sept. 1989.

[18] K. L. Calvert and S. S. Lam, "Adaptors for Protocol Conversion," in *Proc. IEEE INFOCOM 1990*, (San Francisco), pp. 552–560, June 1990.

[19] K. L. Calvert and S. S. Lam, "Formal Methods for Protocol Conversion," *IEEE Journal on Selected Areas in Communications*, vol. 8, pp. 127–142, Jan. 1990.

[20] K. Okumura, "Generation of Proper Adaptors and Converters from a Formal Service Specification," in *Proc. IEEE INFOCOM 1990*, (San Francisco), pp. 564–571, Jan. 1990.

[21] Y.-W. Yao and M. T. Liu, "Constructing Protocol Converters from Service Specification," in *Proc. IEEE ICDCS-12*, (Yokohama, Japan), pp. 344–351, June 1992.

[22] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Language and Computation.* Reading, Massachusetts: Addison-Wesley Publishing Company, 1979.

Appendix

Outline of the Algorithm

Step 1. *Generate the Service System Graph G*:
The service system graph is obtained by applying the modified composition operation, $\otimes$, (defined in Definition 4) to the given service specifications, P_s and Q_s and the conversion service requirement R, where P_s, Q_s and R are as defined in Definition 3 (the Protocol Conversion Problem). In short, G is obtained by

$$G =< P_s, R, Q_s >= P_s \otimes R \otimes Q_s$$

Step 2. *Derive the Conversion Service Specification M*:
M is derived by applying the projection operation, $|$ (defined in Section 3.2), to G as follows.

$$M = G \mid_{\Sigma_{P_b} \cup \Sigma_{Q_a}}$$

where Σ_{P_b} and Σ_{Q_a} are the service transition sets of P_b and Q_a, respectively.

Step 3. *Generate the Synchronizing Transition Sets*:
The synchronizing transition sets are generated from the conversion service specification M obtained in Step 2.

(a) Generate all *legal* paths of M, where a legal path, p_i, is a sequence of service transitions that is accepted by M. In other words, a legal path is a sequence of service transitions which starts from the initial state of M and ends also at the initial state. (Since M is a deterministic CFSM, the algorithm described in [22] can be used to convert the CFSM into a set of regular expressions. From the regular expressions obtained all the legal paths can be easily generated) For each path generated, check if it contains transitions from only one entity (i.e. either from P or Q but not both). If so, the transitions in the path have nothing to do with synchronization between protocols P and Q and can be removed from M and from all the other paths already found. In addition, these service transitions can also be removed from the original protocol specification (P_b and/or Q_a) to reduce the complexity at Step 4 of the proposed algorithm. This step stops when *all* the relative execution orders of the transitions in M are represented by at least one legal path generated.

(b) For each path, p_i, generated in (a), create a transition set, T_i, whose members are the transitions in the path.

$$T_i = \{ \text{ t } \mid \text{ t is a transition in } p_i \}$$

(c) For each transition set generated in (b), use the following definition to identify if it is a synchronizing transition set S_k, where k is an arbitrary number used to distinguish one synchronizing transition set from another.

Definition Synchronizing Transition Set S:
A transition set T_i is a synchronizing transition set if the following is true:

$$\exists \text{ j} \neq \text{i} \ni T_i = T_j$$

& p_i and p_j are two different permutation of each other.

Note that T_i, T_j, p_i and p_j are as generated in Step (a) and Step (b) above. The synchronizing transition set, S_k, represents a set of service transitions that must be synchronized between protocols P and Q to perform the functions requested by conversion service requirement R.

Step 4. *Compose the Converter using the Synchronizing Transition Sets*:
For each synchronizing transition set S_k found in Step 3, create a virtual atomic transition V_k. After all such V_k's (from all S_k's obtained in (c) above) are found, the converter specification is then derived by applying the refined coordinated composition operation, $\odot$, defined in Definition 7, to the protocol specifications P_b and Q_a and the V_k's created:

$$\text{converter } C = P_b \odot Q_a$$

Step 5. *Remove the service transitions*:
From the converter specification C obtained at Step 4, change all the service transitions into null transitions and then use the algorithm described in [22] to remove them from C. If a virtual atomic transition introduced at Step 4 corresponds to only some service transitions (as in Example 1), it can also be removed immediately; otherwise, it must be expanded into a sequence of peer transitions (as in Example 2).

B. H. W. Jeng and M. T. Liu,
"Protocol Conversion in Multimedia Networks: Simulation and Algorithm,"
Simulation,
Vol. 64, No. 1, pp. 51–60, January 1995.

SIMULATION 64:1, 51-60
© 1995, Simulation Councils, Inc.
ISSN 0037-5497/94 $3.00 + .10
Printed in the United States of America

Protocol Conversion in Multimedia Networks: Simulation and Algorithm

Hou-Wa J. Jeng, and Ming T. Liu
Department of Computer and Information Science
The Ohio State University
2036 Neil Avenue
Columbus, OH 43210-1277
Tel: (614) 860-5955
Fax: (614) 868-2002
E-mail: jeng@cis.ohio-state.edu

Out of recently proposed high speed networks such as DQDB, FDDI, ATM, etc., various issues have been examined to determine the most suitable type of backbone network for multimedia networks[1]. At this point in time, evaluations are still inconclusive. As a result, different high speed networks using different protocols will co-exist in an integrated multimedia network in the near future. Like the traditional heterogeneous data networks, the inter-operability among the interconnected heterogeneous high speed networks will be the key to success for integrated multimedia networks. On the other hand, unlike the traditional heterogeneous data networks, a simulation study presented in this paper shows that, due to the special characteristics (e.g., high speed and high connectivity) of these integrated multimedia networks, the traditional protocol conversion model will introduce long transmission delays and render a network system useless. Thus, in addition to the simulation study, efforts are made in this paper to modify the previously proposed Synchronizing Transition Set (STS)[2] algorithm to accommodate the high speed and high connectivity of a multimedia network. For the modified algorithm, the protocol compensation *model is used instead of the traditional intermediate converter model while all the desirable strengths of the original STS approach are retained.*

Keywords : multimedia network, simulation, protocol conversion, service specification, synchronizing transition set

1. Introduction

In recent years, rapid advance in network technologies and transmission infrastructure, using optical fiber, has resulted in an unprecedented telecommunication exploration of integrated multimedia networks. Technologies in many computer science areas are being explored to support the ever growing demands of multimedia applications [3,4,5,6,7,8]. In the mean time, a variety of high speed networks, such as ATM, FDDI, and DQDB, which exhibit speeds in the range of hundreds of Mbits/s and even Gbits/s, have been proposed as the potential backbone networks for supporting multimedia applications. So far, there is not one network from the proposed lists which is superior to all the others in all aspects. For example, as pointed out in [1], to achieve better resource utilization, the bandwidth allocation at a multiplexing point in an ATM network is dynamic. This in turn facilitates support of bursty traffic types of multimedia applications. On the other hand, an FDDI network could be favored for non-bursty transmission of voice and video packets due to its guarantees of bounded access delay. Yet under light traffic load, a DQDB network is favored for its low access delays to asynchronous traffic. Moreover, the FDDI network, which is based on token ring topology,

Research reported herein was supported by U.S. Army Research Office, under contract No. DAAL03-92-G-0184. The views, opinions, and/or findings contained in this paper are those of the authors and should not be construed as an official Department of the Army position, policy or decision.

is best suited for local area networks, while the DQDB network, which has bus topology, has been adopted by the IEEE 802.6 Working Group as a standard for metropolitan area networks. On the other hand, ATM networks are best suited for high-speed wide area networks. Because different high speed networks pose different limitations on the range of coverage, and exhibit different characteristics (e.g., access delay, bandwidth allocation), in the foreseeable future, diverse multimedia applications will require co-existence of various high speed networks in an integrated multimedia network.

To achieve interoperability, many researchers have started to study aspects of heterogeneous high speed networks interconnections [9,10,11,12,13,14,15]. However, all the proposed solutions to date are ad-hoc approaches which cannot be applied to high speed network systems other than their original targets. Consequently, they are expensive and limited. In this paper, the previously proposed formal Synchronizing Transition Set (STS) algorithm[2] is modified in an attempt to solve the protocol conversion problem cost-effectively in multimedia networks.

Nevertheless, protocol conversion in multimedia networks will face more challenging issues than in traditional data networks. For example, it has been projected that a single multimedia network could be used to connect every HDTV, video/audio phone, and work-station in a wide area. Upon successful development of a multimedia network, each household of the covered area will be able to simply plug their HDTV sets into the network to receive TV signals. Similarly, after plugging in a video phone, video phone owners will be able to reach others just by dialing the destination numbers. In this scenario, the number of nodes connected in the multimedia networks is at least the number of households in the covered area. As a result, the connectivity (number of nodes) of a multimedia network must be much higher than a traditional data network. Therefore, in addition to high speed, high connectivity becomes another dominant characteristic of a multimedia network that must be considered in protocol conversion. Consequently, the traditional protocol conversion model (Fig. 1) is no longer feasible, as shown in the simulation study presented in Section 2. Thus, in contrast to the traditional conversion model, a model that performs conversion at end nodes is proposed in this paper. This category of conversion is known as the *protocol compensation* approach.

In the remainder of this paper: Section 2 presents the simulation results that demonstrate the inappropriateness of the traditional conversion model in multimedia networks; Section 3 describes the protocol compensation model; in Section 4, an example from [2] is used to demonstrate the modified STS algorithm step by step; Section 5 presents a refinement of the algorithm for reducing the complexity of converter composition at step 4; and in Section 6, conclusions are presented.

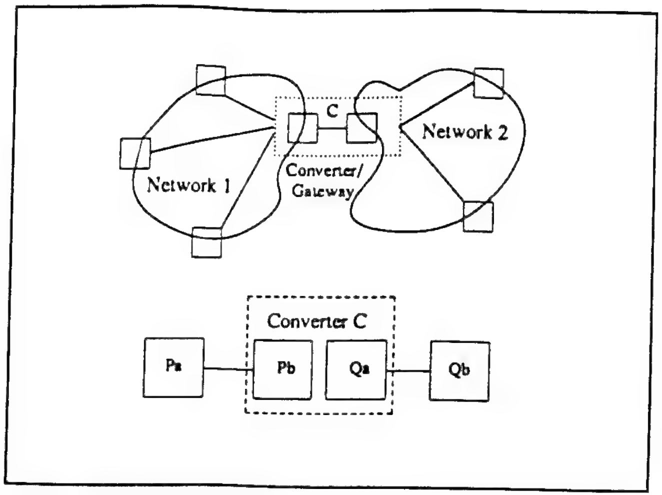


Figure 1. Traditional conversion model

2. Simulation

In this section, simulation results are presented to demonstrate that the message delays introduced by a traditional intermediate node converter will make multimedia network gateway a traffic bottleneck. The model to be simulated is as shown in Fig. 1, in which multiple nodes in one network share the same converter gateway to communicate with multiple nodes in another network. For ease of simulation, all sessions that cross through the converter are assumed to have the same characteristics.

In general, a Monochrome-Video has a window size of 512 x 256 pixels. Assuming each pixel is represented by one byte of data, a video frame would thus generate 128K bytes of data, which is 1 Mega-bits/per-frame. For continuous playback of a video program, a rate of approximately 30 frames per second is necessary. Therefore, to support one continuous video playback multimedia session, approximately 30 Mbps of bandwidth is consumed. However, since it is unlikely that a session would perform such costly operation all the time, the actual probability p that a session has a video frame ready for transmission at a given unit of time (ms in the simulation) varies during the life time of the session. The maximum value of this probability p is 0.03 (= 30 frames / 1000 ms) for continuous video playback, and it has lower value for other types of applications. In the simulations presented, different values of p are used. Note that, for the ease of simulation, when a value of p is chosen for a simulation run, its value remains the same during the run; also, all sessions during the run will have the same value of p.

Currently, a typical high speed network can transmit data in the range of a few hundred Mbits/s to some Gbits/s. To match the high speed, very high throughput processors must be used at converter gateways. Note that even with the highest power processors, the maximum converter throughput will be much lower than the full bandwidth of the backbone network due to the latency introduced by the message buffering and conversion processing at converter gateways. In the presented simulations, converters with different throughputs in the range of 500 Mbits/s to 1.5 Gbits/s are considered, and for the simplicity of simulations, the propagation delay is ignored.

In short, the simulations are conducted as follows. Frames ready to transmit at a source node will arrive at a converter immediately. A converter delivers frames in the order of their arrivals (as a FIFO queue). In a unit of time (1 ms), a converter can deliver at most an amount of data (or a number of video frames) equal to its maximum throughput; i.e. depending upon the maximum throughput of the converter and the number of frames arrived, a converter may deliver only part of them in a unit of time. If the remaining throughput at a given unit of time is not enough to transmit a complete frame, a partial frame can be delivered; the rest of the partially delivered frame will have higher priority for delivery in the next unit of time. Therefore, due to the limited throughput of a converter, those frames that cannot be delivered immediately are put into the converter's buffers. Again, for the sake of simplicity, it is assumed that all converters have unlimited buffers such that frames are never discarded at a converter. In such model, the transmission delay of a frame is the period between the time it arrives at a converter and the time it leaves for its destination (note that, this is true only because of the assumption of no propagation delays in the simulations). During a simulation run, the transmission delay encountered by a frame is accumulated into a total frame delay (in ms). This accumulated delay is used to calculate the average frame delay, which is obtained by dividing the accumulated delays by the total number of frames transmitted.

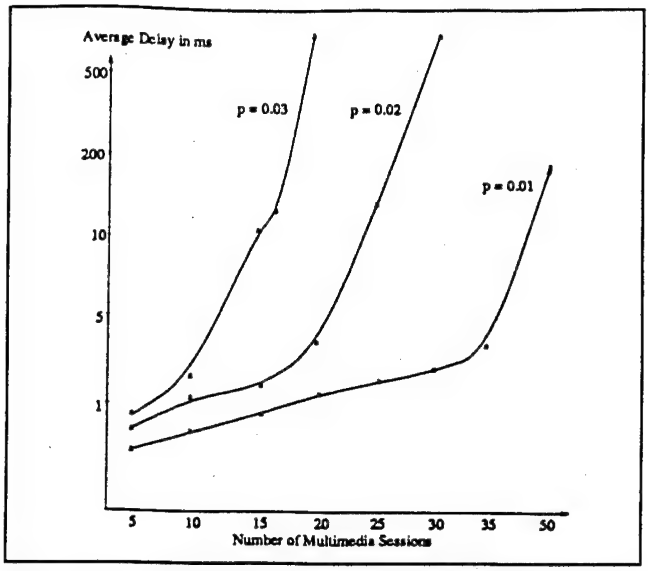


Figure 2. Simulation with Converter of 500 Mbits/s Maximum Throughput

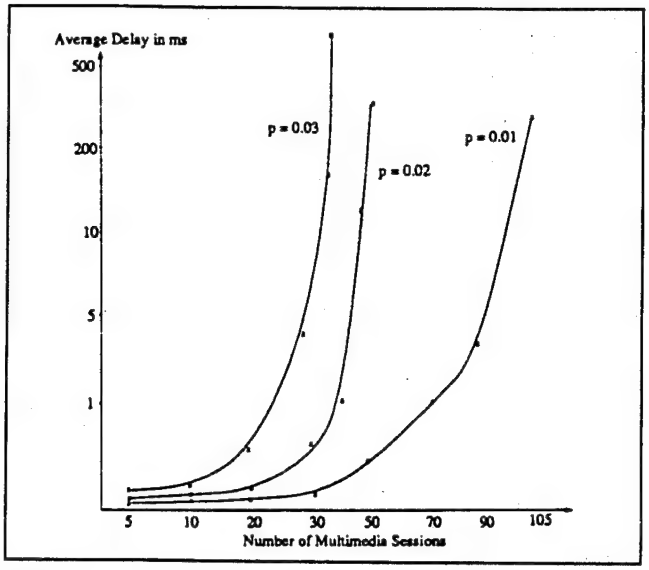


Figure 3. Simulation with converter 1 Gbits/s Maximum Throughput

Fig. 2, in which the converter maximum throughput is 500 Mbits/s, shows the relationship between the average frame delay and the number of active multimedia sessions under different values of frame generation probability p. The same simulation is also run against converters with maximum throughputs of 1 Gbits/s and 1.5 Gbits/s, and the result is shown in Fig. 3 and 4, respectively. The actual numbers obtained from these simulations are listed in Table 1,2 and 3, respectively. Note that, under the assumptions of these simulations, using 1 converter gateway of 1.5 Gbits/s throughput is equivalent to using simultaneously 3 converter gateways of 500 Mbits/s throughput each.

From the results of these simulations, it can be verified that the average frame delays are affected the most by the number of active multimedia sessions and the values of the frame generation probability p. The average frame delays grow almost exponentially as the number of sessions increases, and the probability p affects the growth rates of the average frame delays: the higher value the probability p is, the faster the average frame delays grow. In fact, as the number of sessions increases steadily, the converters quickly reach their

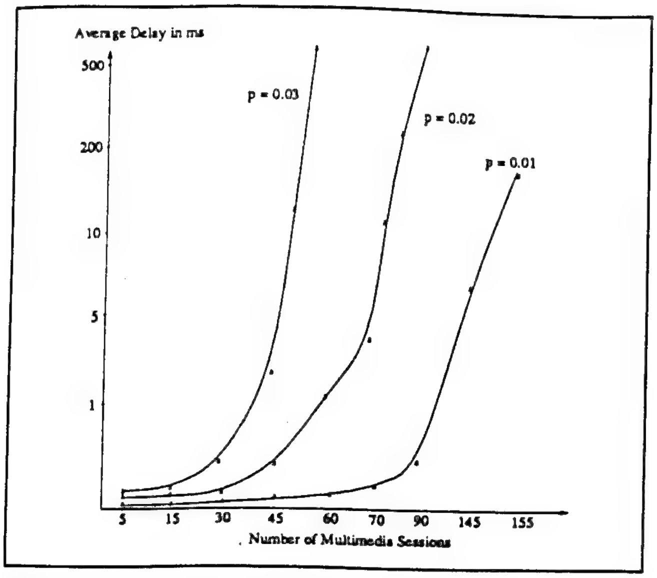


Figure 4. Simulation with Converter of 1.5 Gbits/s Maximum Throughput

maximum throughputs and the frame delays become unmanageable. Note that for a typical multimedia application of continuous video playback of 30 frames/s, the maximum acceptable frame delay is 33 ms. In the traditional intermediate converter model, to keep the average frame delay under 33 ms, a high performance converter of 1 Gbits/s throughput can effectively support less than 100 sessions only. Even though increasing the power or the number of the converters can reduce the bottleneck effect, it can never keep up with the increase in number of sessions in an integrated multimedia network with high connectivity. Stil these gateways would be points of traffic congestion and delay (as shown by Table 2, and 3).

Table 1. Simulation with 500 Mbits/s Throughput Converter

Simulation Number	Frame Generation Probability p	total # of Sessions	Average Delays (ms)	Used Converter Throughput (%)
1.	0.010	5	0.607	10
2.	0.010	10	0.748	20
3.	0.010	15	0.901	30
4.	0.010	20	1.293	41
5.	0.010	25	1.424	49
6.	0.010	30	1.951	59
7.	0.010	35	2.506	68
8.	0.010	40	4.303	78
9.	0.010	45	6.661	88
10.	0.010	50	194.840	100
11.	0.010	55	610.080	100
12.	0.020	5	0.728	20
13.	0.020	10	1.146	41
14.	0.020	15	1.967	62
15.	0.020	20	4.131	80
16.	0.020	25	77.055	99
17.	0.020	30	936.981	100
18.	0.020	35	1448.063	100
19.	0.030	5	0.926	31
20.	0.030	7	1.124	41
21.	0.030	9	1.616	55
22.	0.030	11	2.372	66
23.	0.030	13	3.963	79
24.	0.030	15	10.118	91
25.	0.030	17	139.942	100
26.	0.030	19	607.542	100

The bottleneck effect is more tolerable in traditional data networks, since the number of their internal nodes is, in general, less than a few hundred and internetwork communication is relatively less frequent. In fact, the throughput requirement of traditional internetwork communication is in the range of hundreds of Kbits/s or less; therefore, in terms of network interconnection, a high speed bridge would suffice. In the case of multimedia networks, however, high connectivity means increased probability of traffic through the converter/gateway. Coupled with the high bandwidth requirements of each media stream, the transmission delay at a single point of traffic congestion becomes unmanageable, as illustrated by the simulations. Clearly, the traditional intermediate node conversion model should be avoided in a high connectivity multimedia network, and a more efficient approach is needed.

3. New Model

To avoid bottleneck effects, the new model eliminates the intermediate conversion node and by contrast, conversion is performed at either of the end nodes as shown in Fig. 5 and Fig. 6. In the new model, it is assumed that a node in one network can physically connect to another network by simply *tapping* itself into an access point of the target network, in the same manner that plugging in a telephone establishes a connection.

In Fig. 5, node N2 in network 2 taps itself into network 1 to initiate internetwork communication with node N1 in network 1. The dotted lines between node N2 and network 1 denote the attachment of N2 to network 1. From node N1's viewpoint, node N2 is only a newly added node in network 1 running the same

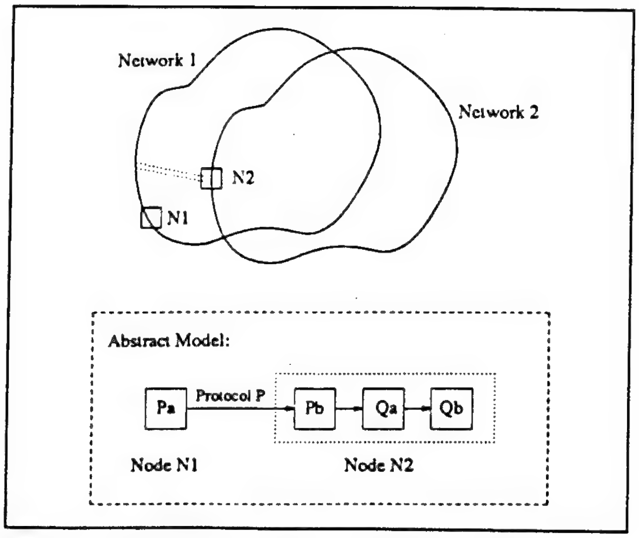


Figure 5. Option 1 - Conversion at receiving node

Table 2. Simulation with Gbit/s Throughput Converter

Simulation Number	Frame Generation Probability p	total # of Sessions	Average Delays (ms)	Used Converter Throughput (%)
1.	0.010	5	0.027	5
2.	0.010	10	0.045	10
3.	0.010	15	0.081	15
4.	0.010	20	0.138	20
5.	0.010	25	0.148	24
6.	0.010	30	0.199	29
7.	0.010	35	0.247	34
8.	0.010	40	0.324	39
9.	0.010	45	0.367	44
10.	0.010	50	0.546	52
11.	0.010	55	0.643	56
12.	0.010	60	0.730	61
13.	0.010	65	0.917	65
14.	0.010	70	1.088	70
15.	0.010	75	1.647	74
16.	0.010	80	1.903	80
17.	0.010	85	2.627	84
18.	0.010	90	4.493	91
19.	0.010	95	11.094	96
20.	0.010	100	26.861	99
21.	0.010	105	309.322	100
22.	0.010	110	520.775	100
23.	0.010	115	715.404	100
24.	0.020	5	0.048	10
25.	0.020	10	0.117	19
26.	,0.020	15	0.175	30
27.	0.020	20	0.298	39
28.	0.020	25	0.489	50
29.	0.020	30	0.816	60
30.	0.020	35	1.017	69
31.	0.020	40	1.662	79
32.	0.020	45	3.661	89
33.	0.020	50	81.516	99
34.	0.020	55	358.539	100
35.	0.020	60	811.540	100
36.	0.030	5	0.059	15
37.	0.030	10	0.195	30
38.	0.030	15	0.395	46
39.	0.030	20	0.670	59
40.	0.030	25	1.502	75
41.	0.030	30	4.326	91
42.	0.030	35	182.920	100
43.	0.030	40	804.266	100

Table 3. Simulation with 1.5 Gbits/s Throughput Converter

Simulation Number	Frame Generation Probability p	total # of Sessions	Average Delays (ms)	Used Converter Throughput (%)
1.	0.010	5	0.012	3
2.	0.010	15	0.037	10
3.	0.010	25	0.070	17
4.	0.010	35	0.096	23
5.	0.010	45	0.133	29
6.	0.010	55	0.178	36
7.	0.010	65	0.235	43
8.	0.010	75	0.358	52
9.	0.010	85	0.408	57
10.	0.010	95	0.546	64
11.	0.010	105	0.800	69
12.	0.010	115	1.014	77
13.	0.010	125	1.630	83
14.	0.010	135	3.174	91
15.	0.010	145	7.783	97
16.	0.010	155	189.726	100
17.	0.010	165	465.497	100
18.	0.010	175	706.039	100
19.	0.020	5	0.018	7
20.	0.020	10	0.043	13
21.	0.020	15	0.078	20
22.	0.020	20	0.110	26
23.	0.020	25	0.150	33
24.	0.020	30	0.206	40
25.	0.020	35	0.286	46
26.	0.020	40	0.358	54
27.	0.020	45	0.445	60
28.	0.020	50	0.609	66
29.	0.020	55	0.840	73
30.	0.020	60	1.218	80
31.	0.020	65	1.861	86
32.	0.020	70	4.489	93
33.	0.020	75	61.049	100
34.	0.020	80	300.565	100
35.	0.020	85	641.509	100
36.	0.020	90	755.993	100
37.	0.020	95	994.235	100
38.	0.030	5	0.034	10
39.	0.030	10	0.077	20
40.	0.030	15	0.124	31
41.	0.030	20	0.204	40
42.	0.030	25	0.283	49
43.	0.030	30	0.461	60
44.	0.030	35	0.716	69
45.	0.030	40	1.222	80
46.	0.030	45	2.774	90
47.	0.030	50	63.673	100
48.	0.030	55	475.614	100
49.	0.030	60	787.235	100
50.	0.030	65	1193.620	100

protocol. N1 and N2 can be considered as two nodes connected by the same high speed links of network 1. Therefore, N1 would use the regular protocol of network 1 to communicate with N2 without knowing that N2 is a node in network 2. To keep the conversion transparent to high level application programs, node N2 performs the conversion within its low level protocol entities. In the dashed box of Fig. 5. protocol P (of network 1) is used between nodes N1 and N2. The transparency property requirement defined in[2] is relaxed in the sense that the conversion is not transparent to the low level protocol entities of the receiving node. Still, the conversion is transparent to node N1 and the high level application programs of node N2. Similarly, when N1 in network 1 is the node which initiates the internetwork communication, it can also attach itself to network 2 as shown in Fig. 6.

In this *protocol compensation* model, nodes initiating internetwork communication perform the protocol conversion. Therefore, the conversion function is decentralized and bottlenecks are eliminated. Consequently, high speed internetwork communication becomes more plausible. Moreover, removing the possible single point of failure that leads to nodal isolation makes this model more fault tolerant.

Despite the aforementioned advantages, a converter gateway still must be used when the networks to be integrated are geographically separated far apart, and tapping into other networks is not feasible. However, it should also be noted that the networks in a multimedia network environment are often overlapped, much like telephone networks and cable TV networks: an average household often contains both cable TV and telephone outlets. Two networks are considered to be overlapping if they cover approximately the same geographical area, and an access point of one network is always close to an

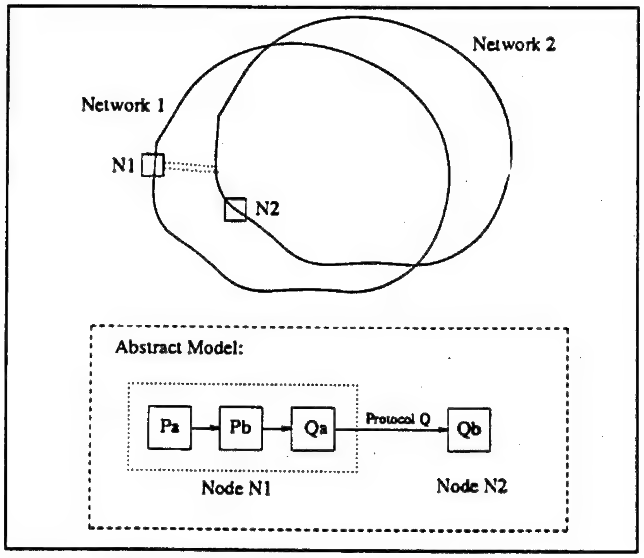


Figure 6. Option 2 - Conversion at Sending Node

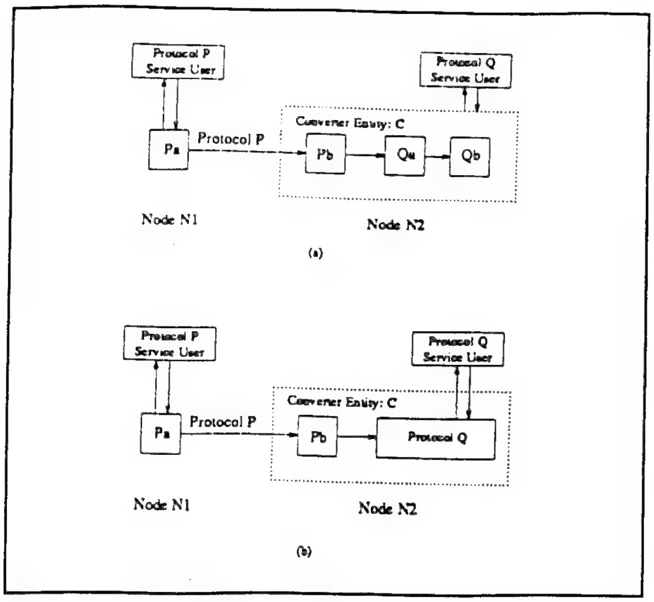


Figure 7. Abstract model of Option 1.

access point of the other. In this sense, if a node needs an extraordinarily long distance link (e.g., from the East Coast to the West Coast of the United States) for tapping into the other network, the tapping is considered unreasonable and traditional gateways must be used.

The remainder of this paper focuses on deriving converters in option 1, and converters in option 2 can be derived similarly. In the service abstraction of option 1 shown in Fig. 7, the conversion is performed by the converter entity C, which supports the service interface of protocol Q, while at the same time using protocol P to communicate with its peer entity P_u. Note that protocol entities P_b, Q_u and Q_b are all combined into a single entity C. Since protocol Q's peer interface (communication links between Q_u and Q_b) no longer exists in the new model, the peer transitions of protocol Q, which were used to support its peer interface, need not be used in the derivation of C. Therefore, the abstraction in part (b) of Fig. 7 becomes the actual model used in the paper. Consequently, it is the service specification Q_s that participates in the composition of converter C at step 4 of the modified algorithm.

4. The Algorithm

For ease of presentation, the second example from[2] is used to give a step by step demonstration of the modified STS algorithm. Fig. 8 shows the given protocol specifications of P and Q, in which the service transitions are labeled in capital letters. Protocol P is a simple non-sequence protocol. Protocol Q is a simplified flow control protocol. Before sending each data packet from Q_u to Q_b, a reservation needs to be confirmed. If Q_b has no space available to receive data, it can send a choke message to Q_u to stop the transmission. The service specifications, P_s, Q_s, and the conversion requirement R are as shown in Fig. 9.

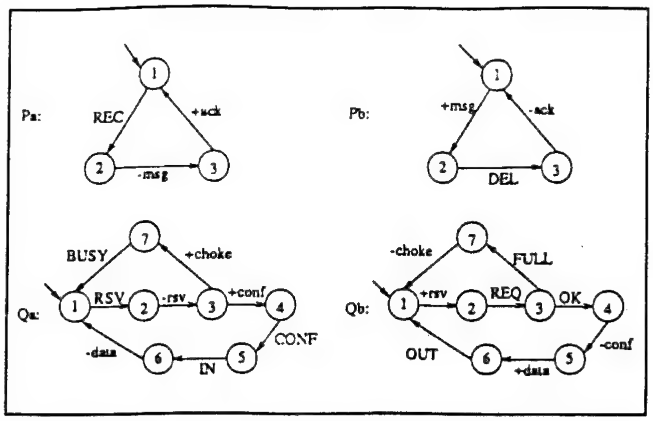


Figure 8. Example - Given Protocols P and Q

In the STS approach, the first 3 steps of the algorithm are used to derive the inter-relationship of the service transitions at the protocol boundary, between the service transitions of P_b and Q_u. In the new model, the conversion is performed via the coordination of service transitions also at the boundary between P_b and protocol Q (specifically, also between P_b and Q_u.) Therefore, the first 3 steps remain the same. The result of applying ⊗ (defined in [2]) to P_s, R and Q_s to obtain the service system graph G is shown in Fig. 10.

The initial projection of G onto the service transitions of P_b and Q_u is shown in Fig. 11, in which states [1, 1, 2] and [2, 3, 3] can be removed along with the incidental null transitions as described in [2]. The result is shown in Fig. 12, in which state [2, 2, 1] (also [1, 2,7] has incoming null and non-null transitions. This situation does not meet the simple conditions described in[2] for the aforementioned state removal. In this case, these states must not be excluded from the FSM; only the incidental null transitions can be removed. To do so, the null transition removal procedure from[2] is revised as follows.

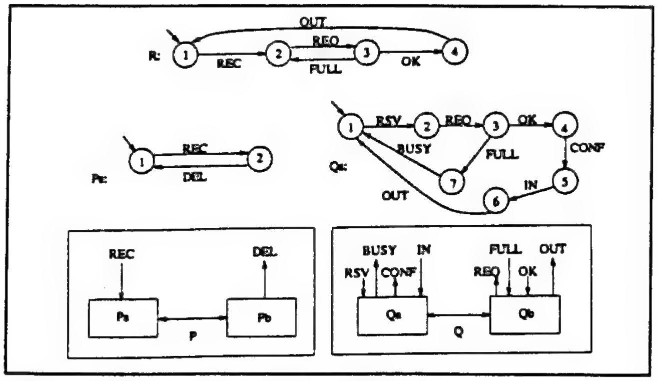


Figure 9. Example - Service Specifications

Revised Null Transition Removal Procedure

1.For each NULL transition a, if a is the only outgoing transition of a state st, st can be removed along with a by redirecting the incoming transitions of st to the tail state of a [2].
2. For each remaining NULL transition a, if a is the only incoming transition of a state st, st can be removed along with a by changing the head states of all of st's outgoing transitions to the head state of a [2].
3. For each non-NULL transition T, whose head and tail states are s and e, respectively, where e is the head state of a NULL transition a, add a new transition T from state s to the tail state of a.
4. For each non-NULL transition T whose head and tail states are s and e, respectively, where s is the tail state of a NULL transition a, add a new transition T from the head state of a to state e.
5. Remove all the remaining null transitions. The resulting FSM is equivalent to the original one.

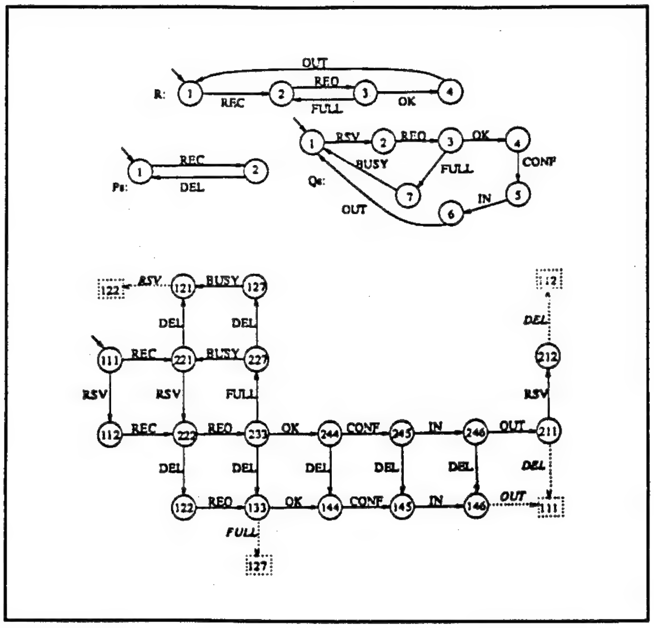


Figure 10. Example - Service System Graph G

After applying the above procedure, the conversion service specification obtained is shown in Fig.13. Next in step 3, a synchronizing transition set is obtained as follows:

virtual atomic transition $V1$ with
synchronizing transition set S_1
= { DEL, RSV, CONF, IN }

The next step (step 4) of the algorithm is the major difference of the new model: the converter entity C in part (b) of Fig.7 is composed of peer protocol entity P_b and the entire protocol Q (as a single entity). In fact, protocol Q is treated as an entity consisting only of service transitions; i.e. the participants of this step in the new model are P_b and Q_s instead of P_b and Q_a. The same coordinated composition operator O defined in[2] is used to derive protocol entity C:

$$\text{converter entity } C = P_b \, \mathrm{O} \, Q_s$$

The result of applying the coordinated composition operator to P_b and Q_s is shown in Fig.14. One may argue that Q_s contains service transitions from Q_a which are not needed in the final converter specification and should not participate in the composition either. In answer to this argument, the service transitions from Q_a are needed for coordinating purposes in the coordinated composition operation. The service transitions of Q_b are synchronized with the peer transitions of P_b by the coordination of the service transitions from the synchronizing transition set which includes service transitions from P_b and Q_a. In short, the service transitions from Q_a are needed to serve as the coordinating media in this step.

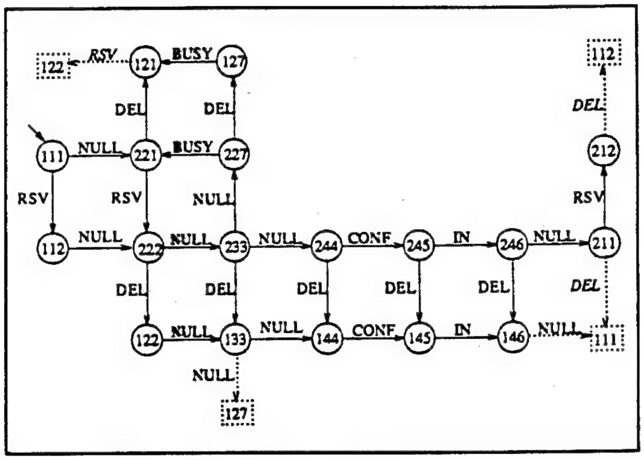


Figure 11. Example - Initial Projection

In the example, note that the synchronizing transition set contains more than one transition from Q_a that have to be executed in a single virtual atomic transition[2]; i.e., RSV, CONF, and IN from Q_a have to be executed together with DEL from P_b. To reduce the complexity of the coordinated composition, Q_s can be simplified by combining the transition sequence

RSV.REQ.OK.CONF.IN

(which includes the service transitions from S_1) into a single atomic transition before the composition with P_b. In Fig.14, Q'_s is the simplified service specification of protocol Q, which in turn participates with P_b in the composition operation.

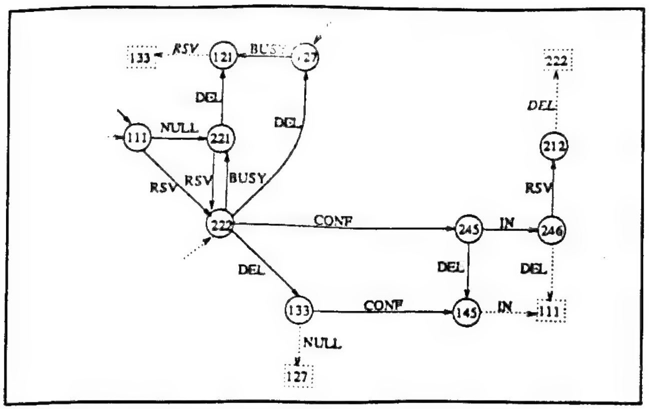


Figure 12. Example - Intermediate Projection Result

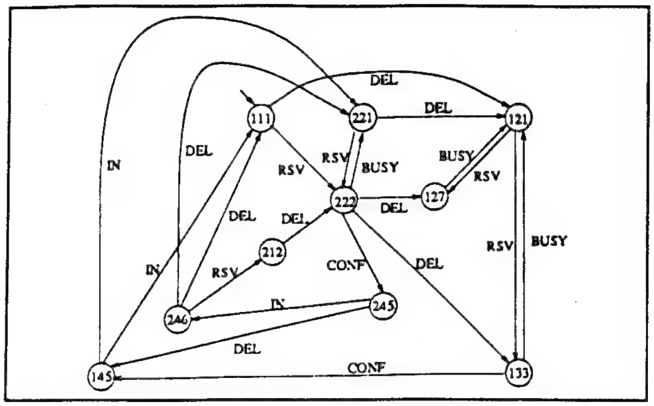


Figure 13. Example - Conversion Service Specification

Finally, as the last step (step 5), the virtual atomic transition V is expanded and the service transitions from Q_a (RSV and BUSY) are removed. The results are shown in Fig. 15 and 16, respectively.

5. Refinement

It can be verified that states [3, 7] and [1, 7] in Fig. 16 are redundant states. This indicates the need for further refinement of the proposed algorithm. Note that these two states are included in the converter entity specification at step 4 when the coordinated composition operator Θ is applied to P_b and Q'_s, which contain service transitions (RSV and BUSY) from Q_a. These service transitions are eventually removed at step 5. In fact, when Q_s was simplified into Q_s', the coordinating capability from RSV and BUSY has been instated into the created atomic transition V'. As a result, RSV and BUSY are no longer needed as coordinating media. Therefore, they can be removed before the composition to reduce the complexity further. In Fig. 17, all Q_a's service transitions are removed when Q_s' is further simplified into Q_s''. Then, a less complex converter is obtained (by applying the $\odot$ operation to P_b and Q_s'').

Note that the converter C in Fig. 17 has no service transitions from Q_a; therefore, there is no need to perform the NULL transition removal after the composition. Again, the final converter is shown in Fig. 18 in which the redundant states [3, 7] and [1, 7] from Fig. 16 are excluded as expected. Formally, step 4 of the revised algorithm is as follows:

Step 4

Let the set of service transitions of Q_a be Σ_{Q_a} and the synchronizing transition sets be S_i, where i = 1 to n, and n is the number of synchronizing transition sets obtained in Step 3. Define sets Θ_{S_i} for i = 1 to n as follows:

$\Theta_{S_i} = \{\theta \mid \theta$ is a subsequence of a legal path of Q_s, $\theta |_{\Sigma_{Q_a}} = S_i |_{\Sigma_{Q_a}}$, and θ starts and ends with transitions from $S_i\}$

In the above definition, $\theta |_{\Sigma_{Q_a}}$ and $S_i |_{\Sigma_{Q_a}}$ denote the projections of θ and S_i onto Σ_{Q_a}, respectively. In the example, for S_1 = { DEL, RSV, CONF, IN }, the corresponding Θ_{S_1} = {RSV.REQ.OK.CONF.IN}. In the following procedures, let $\theta_j^{\Theta_{S_i}}$ denote a member of Θ_{S_i}, where j = 1 to $|\Theta_{S_i}|$:

1. Create an atomic transition $V_j^{\Theta_{S_i}}$ for each $\theta_j^{\Theta_{S_i}}$. In Q_s, replace each $\theta_j^{\Theta_{S_i}}$ with the corresponding $V_j^{\Theta_{S_i}}$ to simplify Q_s. The execution of $V_j^{\Theta_{S_i}}$ represents the execution of all transitions of $\theta_j^{\Theta_{S_i}}$ in an atomic manner. Denote the resulting FSM as Q'_s. In the example, $V_1^{\Theta_{S_1}}$ and Q'_s are V' and Q'_s, respectively, as shown in Fig. 14 and Fig. 17.

2. Use the null transition removal algorithm to remove all remaining service transitions belonging to Q_a from Q'_s. Denote the resulting FSM as Q''_s.

3. Apply the coordinated composition operation $\odot$ to P_b and Q''_s to derive the converter entity C.

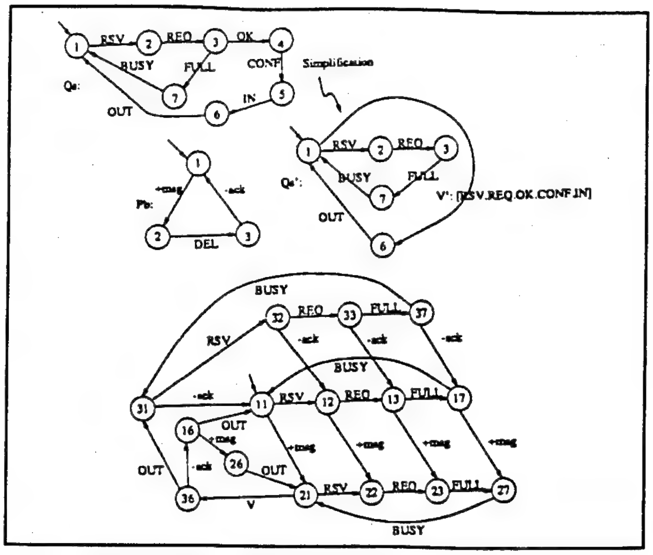


Figure 14. Example - Converter Entity C.

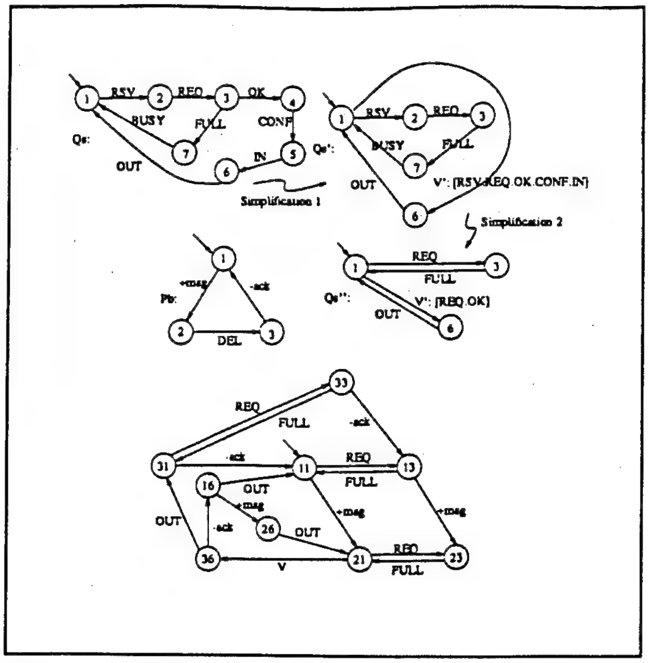


Figure 17. Example - Refinement of Converter.

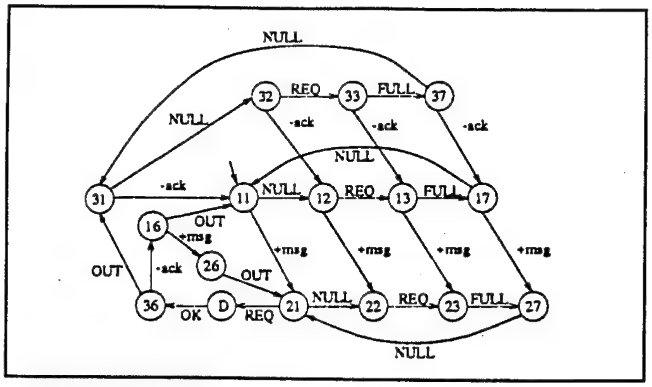


Figure 15. Example - Converter Entity C (V Expanded).

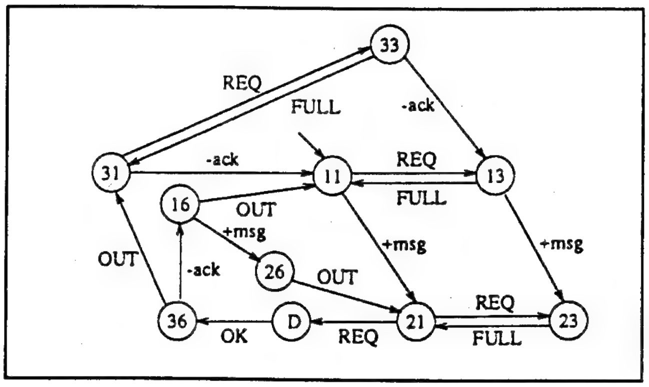


Figure 18. Example - Refined Converter with V Expanded.

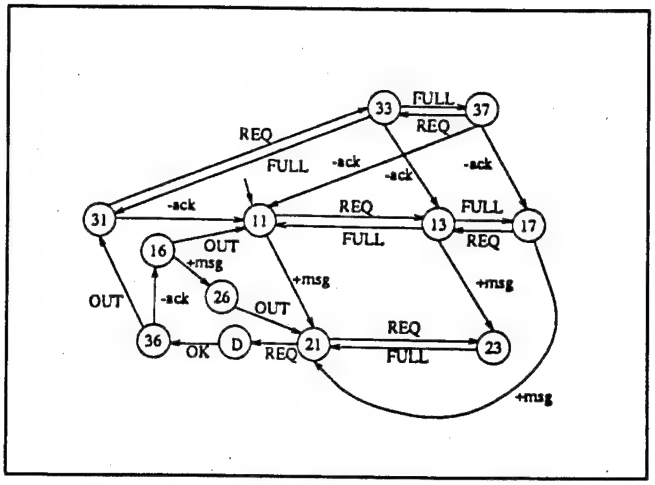


Figure 16. Example - Converter Entity C with NULL Transitions Removed.

6. Conclusion

In this paper, a simulation study is presented to show the shortcomings of the traditional protocol conversion model in multimedia networks.At the conclusion of the simulation study,it is shown that a more efficient conversion model is needed.Thus, to achieve better efficiency, a protocol compensation model is proposed to perform protocol conversion in multimedia networks. Under this protocol compensation model, the previously proposed Synchronizing Transition Set (STS) algorithm is modified to derive converter entity at end nodes in an overlapping multimedia network environment.

Since the same ⊗, ⊙ and | operations are used as before, a formal proof of correctness for the modified algorithm can be constructed in the same manner as described in [2]. Moreover, all the beneficial characteristics of the original STS approach are inherited in the modified algorithm, with the exception of the transparency property at a lower layer of the target protocol.These characteristics include, but are not limited to less complexity by using the service specification approach; capability of converting a sequence of transitions as a unit; liveness/conformity properties; and correctness without the need for final validation phase.As emerging high speed networks continue to mature, it is believed that protocol conversion will play an important role in developing a successful integrated multimedia network.

Acknowledgement

The authors would like to thank Mr. Thomas T. Humphrey for his reviewing and valuable comments during the final stages of this work.

References

1] S. Dupuy, W. Tawbi, E. Horlait, "Protocols for High-Speed Multimedia Communications Networks," *Computer Communications*, vol. 15, pp. 349-358, July 1992.

[2] H. J. Jeng and M. T. Liu, "From Service Specification to Protocol Converter: A Synchronizing Transition Set Approach," *Computer Networks and ISDN Systems*, 1992.

[3] P. Rangan, H. Vin, S. Ramanathm, "Communication Architectures and Algorithms for Media Mixing in Multimedia Conferences," *IEEE/ACM Transactions on Networking*, vol. 1, pp. 20-30, Feb. 1993.

[4] C. Turner and L. Peterson, "Image Transfer: An End-to-End Design," in *Proc. SIGCOMM 92*, (Baltimore, Maryland), pp. 258-268, Aug. 1992.

[5] P. Hoepner, "Synchronizing the Presentation of Multimedia Objects" *Computer Communications*, vol. 15, pp.557-564, Nov. 1992.

[6] J. Gemmell and S. Christodoulakis, "Principles of Delay-Sensitive Multimedia Data Storage and Retrieval," *ACM Transactions on Information Systems*, vol. 10, pp. 51-90, Jan. 1992.

[7] E. Moeller, A. Scheller, G. Schurmann, "Distributed Multimedia Information Handling," *Computer Communications*, vol. 13, pp. 232-242, May 1990.

[8] S. Ramanathm and P. Rangan, "Adaptive Feedback Techniques for Synchronized Multimedia Retrieval over Integrated Networks," *IEEE/ACM Transactions on Networking*, vol. 1, pp. 246-260, Feb. 1993.

[9] P. Crocetti, L. Fratta, M. Gerla, M. Marsiglia, D. Romano, "Interconnection of LAN/MANs through SMDS on Top of an ATM Network," *Computer Communications*, vol. 16, pp. 83-92, Feb. 1993.

[10] W. Schodl, U. Briem, H. Kroner, T. Theimer, "Bandwidth Allocation Mechanism for LAN/MAN In-terworking with an ATM Network," *Computer Communications*, vol. 16, pp. 93-99, Feb. 1993.

[11] G. Clapp, "LAN Interconnection Across SMDS," Media Objects," *Computer Communications*, vol. 15, *IEEE Network Mag.*, vol. 5, pp. 25-32, Sept. 1991. pp. 557-564, Nov. 1992.

[12] T. V. Landegem and R. Peschi, "Managing a Connectionless Virtual Overlay Network on Top of ATM," in *Proc. ICC'91*, (Denver, CO), June 1991.

[13] L. Mongiovi, M. Farrel, V. Trecordi, "A Proposal for Interconnecting FDDI Networks Through B-ISDN," in *Proc. INFOCOM'91*, (Bal Harbour, FL), Apr. 1991.

[14] M. Gerla, T. Tai, J. Monteiro, G. Gallassi, "Interconnecting LANs and MANs to ATM," in *Proc. ICN Conf.*, (Minneapolis), Oct. 1991.

[15]A. Biocca, G. Freschi, A. Forcina, R. Melen, "Architectural Issues in the Interoperability between MANs and the ATM Network," in *Proc. ISS'90*, (Stockholm, Sweden), May 1990.

HOU-WA J. JENG received the BS degree in Information Science from the National Taiwan University, Taipei, Taiwan in 1981 and the MS degree in Computer and Information Science from the Ohio State University in 1986. Since 1986, he has been with the Bell Laboratories in Columbus, Ohio. Currently, he is a member of technical staff of the Bell Labs and a part time PhD student in the Department of Computer and Information Science at the Ohio State University, Columbus, Ohio. His major research interest is in computer networking and protocol engineering.

MING T. LIU received the MSEE and PhD degrees from the University of Pennsylvania in 1961 and 1964, respectively. Since 1969 he has been with the Ohio State University, where he is presently Professor of Computer and Information Science. He served as Editor and Editor-In-Chief of *IEEE Transactions on Computers* from 1982 to 1990, and as a member of the Board of Governors from 1984 to 1990. Currently he serves as Program Co-Chairman of the 1995 International Conference on Network Protocols and Steering Committee Chairman of the International Conference on Distributed Computing Systems. He was elected an IEEE Fellow in 1983 for his contributions to computer networking and distributed computing. He is also a member of ACM.

C. A. Elsaadany, M. Singhal and M. T. Liu,
"Priority Communication Schemes on Local
Area Networks for Multimedia Traffic,"
Proc. 19th IEEE Conf. on Local Computer Networks,
pp. 372–379, October 1994.

Priority Communication Schemes on Local Area Networks for Multimedia Traffic

Amr Elsaadany, Mukesh Singhal and Ming T. Liu*
Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210

Abstract

In this paper we address the issues related to the delivery of multimedia streams on local area networks. Multimedia integrates voice and video along with text and images into existing systems. Traditionally, local area networks were designed to handle regular data traffic which are bursty in nature and for which variable delay is acceptable. Multimedia traffic, on the other hand, requires constant delay in addition to fast (or real time) delivery. Multimedia traffic also requires large bandwidth compared to regular traffic. Since local area networks are widely in use, their modification to integrate multimedia streams is very important. In this paper we propose priority-based communication schemes for the timely delivery of multimedia traffic on local area networks. We compare the performance of these schemes using simulation techniques.

Key words: Multimedia, local area networks, priority.

1 Introduction

Multimedia is a media that integrates the transmission of voice and video along with text and images into existing systems [1,2,3,4,5]. Digital multimedia transmits multimedia information digitally and allows them to be manipulated and stored on a computer system. It also allows for interactive human interface. Interactive multimedia allows the user to interface with the system in real time which is important for education, business, entertainment, and communications.

A large number of commercial organizations as well as universities use some type of local area network environment. These organizations have two options for adding multimedia capabilities. One option is to utilize the existing networks by adding some devices and/or software in order to transmit multimedia streams. The other option is to use an ATM type network (see section 3.2). ATM is designed such that voice and video can be transmitted effectively along with data packets. ATM switches and interfaces will be needed which would add a lot of cost to these organizations.

Existing local area networks have two main problems with regard to the integration of multimedia streams. First, the existing bandwidth capability for a standard LAN does not support multimedia transmission requirements. Second, there is no explicit priority mechanism for transmitting data/stream on LANs. Multimedia streams must be received at the user/presentation site at a time that there would be no interruption (or gap) from user's point of view (consider a moving picture presentation). Note that multimedia streams are isochronous and require a constant delay between packets.

This paper focuses on the problems associated with supporting multimedia on local network environments. The protocols of the local area networks are not suitable for multimedia traffic. They were designed to handle regular data traffic which are bursty in nature and for which variable delay is acceptable. Traditional communication protocols for local area networks have to be modified to suit the needs of multimedia applications. We present protocols for priority-based communications on Ethernets to allow timely delivery of multimedia traffic on Ethernets and compare the performance of the possible alternatives.

* Research reported herein was partially supported by U.S. Army Research Office, under contracts No. DAAL03-91-G-0093 and No. DAAL03-92-G-0184. The views, opinions, and/or findings contained in this paper are those of the authors and should not be construed as an official Department of the Army position, policy or decision.

0-8186-6680-3/94 $04.00 © 1994 IEEE

The rest of the paper is organized as follows: In the next section, we present data requirements of multimedia systems. In Section 3, we give a brief background of communication networks. Section 4 presents two techniques for priority-based communication on Ethernets. In Section 5, we present a simulation study of the proposed techniques. Finally, Section 6 concludes the paper.

2 Multimedia Data Requirements

All digital media requires digital end-to-end transmission of multimedia data streams. This means that the communication network must support the transmission of digitized voice, video and images. The network must also have a wide bandwidth to support these transmissions. Not all existing networks have sufficient bandwidth to accommodate the needs of all kinds of multimedia applications. For example, a LAN with 10 Mbps bandwidth may be able to provide limited support to transfer video and images for some multimedia applications.

Multimedia applications require fast processing and small visual response time. Moreover the effect of latency in transmitting streams between stations should be minimized or eliminated.

Moreover the amount of storage required by multimedia data is much more than that of regular data. A 640 x 480 24-bit color resolution graphic image, for example, takes about 900KB of memory. Most common audio CD uses 16-bit resolution and 44.1KHz sampling rate. One second of this high quality audio requires 176.4 kB of storage. Vedio is composed of a stream of frames (e.g., NTSC uses 30 per second). One second of 30 fps video of size 640 x 480 pixels and 24 bit color resolution requires about 27MBps bandwidth.

Multimedia data compression techniques reduces the amount of bandwidth required by streams such as video. In addition, the amount of disk storage required for multimedia data can be substantially decreased. The 27MBps transfer rate required for the 640 x 480 screen would be reduced to 550kBps using the MPEG1 standard [6]. MPEG1 can also compress audio by a factor of 5-10.

It is clear that the amount of data needed to represent video digitally places tremendous requirements on storage, transmission, bandwidth as well as display capabilities of current systems. In addition to the storage and bandwidth requirements for multimedia data, there are some other problems associated with multimedia such as the suitability of existing network protocols. More information about the properties of the multimedia data types/streams and the requirements of multimedia applications can be found in [7,8].

3 Communication Network Background

3.1 Local Area Networks

Local area networks (LAN) exist in one of the following architectures; bus, tree, token ring and FDDI (Fiber Dual Data Interface) ring [9,10]. The most widely used LAN is the Ethernet which is a bus-based architecture. This type of network typically has 10 Mbps bandwidth as opposed to 100 Mbps for FDDI.

Local area network bus (non token) protocols are based on broadcasting. The most dominating medium access protocol (MAC) in use is the CSMA/CD (Carrier Sense Multiple Access with Collision Detection) in which each station listens to the channel while transmitting. In case a collision is detected during transmission, the station ceases transmission and retries after some (random) period of time. This protocol requires that the message size be longer than twice the propagation time (in order to detect collision before transmission is complete). It also use binary exponential backoff technique, in which, after each repeated collision, the mean value of the random delay is doubled.

One other MAC that is widely used is the token bus in which stations on the bus form a logical ring with a token (control packet) to regulate access to the medium. When a station receives the token, it get control of the medium for a specified time. When the station is done, or time expires, it passes the token to the next station in the logical ring.

LAN topologies are based on sharing the bandwidth between stations. Data sent are assumed to be time-independent, they can be broken into packets without regard to their order. Also, when a station is using the network for transmission, no other station can and the entire bandwidth is assigned to it.

Options for Multimedia:

To overcome the problems with the speed and/or order of transmission on LAN, several alternatives are being explored. Fast Ethernet is one way to solve the bandwidth problem of the standard Ethernet and it is expected to be 10 times wider. It may retain the CSMA/CD MAC protocol to allow internetworking with standard Ethernet LANs.

Another way to overcome the bandwidth problem is to use switching hubs. Switched Ethernet establishes a single point-to-point LAN between each workstation and a high-capacity switching hub [11]. This gives each workstation its own LAN with all of its bandwidth. (Some topologies may have more than one workstation per LAN.) However the switching hub may be a bottleneck at some point. That is why this network architecture imposes limits on the number of workstations that could be connected to the hub, and also on the number of simultaneous users.

Other options include using FDDI (running over twisted pair for a short distance) to increase the bandwidth to 100 Mbps. FDDI technology could be extended to support the timely delivery required by multimedia applications. Isochronous Ethernet is another option which provides isochronous channels to stations equipped with isoENET cards [12].

3.2 Switching Networks

Switching networks transfer information between source and destination via three functions: transmission, multiplexing, and switching. Transmission provides the point-to-point transfer of information. Multiplexing uses a single transmission facility (link) for several independent channels. Switching uses properties of telecommunications traffic to replace a fully meshed network interconnecting all pairs of users with a shared substructure providing on demand connections, that is, packets can be assinged to channels based on the address information in the header of the packets themselves.

Circuit switching networks use time division multiplexing (TDM) where bandwidth is multiplexed between several channels with assigned amounts of bandwidth. Time slots within a frame are allocated to the channel for the duration of the service. This type of networks is suitable for applications that require guaranteed bandwidth or low latency such as PCM voice and constant bit rate data traffic. The mode of transfer in this type of network is referred to as synchronous transfer mode (STM).

Packet switching networks use bandwidth efficiently to suit bursty applications such as file transfers. However, the delay in these networks is unpredictable and the latency tends to be high. Packet switching uses statistical multiplexing which dynamically allocates time slots to incoming channels based on demand. In contrast with synchronous TDM, the positional significance of the slots (in TDM) is lost and hence each packet has to carry addressing information in addition to data.

ATM (Asynchronous Transfer Mode) uses cell switching which uses bandwidth flexibly and efficiently for all sorts of applications. For constant rate applications, a guaranteed number of cells per second can be allocated and any unused cell slots can be allocated (on demand) to bursty applications. In addition, the number of virtual channels carrying these cells can vary as well as the rate of each channel.

In cell switching technology, a cell is a 53 byte fixed length packet with 5 bytes for header (containing routing and quality of service information) and 48 bytes for data (payload section) [13]. This fixed length cells are better suited for high speed switching because it enables simpler implementation of switches and provides predictable system behavior.

The asynchronous nature of ATM comes from the fact that cells are assigned asynchronously to any service based on its needs. This is facilitated by the fact that each cell is addressed independently to any destination. In this mode of transmission, there is no notion of owning a time slot on periodic basis.

Although existing wide networks (WAN) can be used by some multimedia applications, these WAN-based applications would suffer from existing problems such as the varying network delays. When it become widely available, ATM will be a suitable technology that has the high bandwidth and low latency requirements for many multimedia applications.

4 Priority Schemes

As mentioned before, our work focuses on the problems associated with supporting multimedia services on local network environments. The protocols for regular data transfer on the local area networks are not suitable for multimedia traffic and they have to be modified to suit multimedia applications. These protocols were designed to handle regular data traffic which is bursty in nature and for which variable delay is acceptable. We present priority-based communication protocols for allowing timely delivery of multimedia traffic on local area networks. We focus our attention to the problems associated with supporting multimedia traffic on with local area bus (Ethernet) as opposed to token ring networks [14].

The current medium access bus protocol does not provide priority for transfer of data. Stations on the bus use the CSMA/CD protocol to compete for bus bandwidth. This means packets can indiscriminately collide and get retransmitted. This medium access protocol is suitable for regular data traffic but not for multimedia traffic. We propose two priority schemes that add some sense of priority to data traffic on the bus. They support two levels of priority: high priority (for multimedia or real-time traffic) and low priority (for regular data).

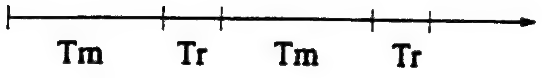


Figure 1. Bus Time

4.1 Bus-Time Multiplexing Priority Scheme

Bus-time multiplexing scheme relies on message passing to support two levels of priority. Before a station sends a "high priority" traffic (for a multimedia session), it informs every other station of its intention so that they would abstain from sending "low priority" traffic for a period of time. This is accomplished by broadcasting a request message that includes the amount of time that the station needs (i.e., the session length). During this time period, only high priority traffic is allowed to use the bus.

Since the request message will compete for the bus with other traffic, the scheme has to ensure that this message gets successfully transmitted before the high priority traffic period starts.

When stations receive a message indicating that high priority traffic need to be sent by a station, they stop sending low priority traffic at once. Stations record the time at which they will be allowed to send regular traffic. This time would be equivalent to the current time plus the length of the session (which is included in the request message). Stations also keep a count of the number of multimedia sessions currently in progress. Whenever a station receives a request message, it increment this count by one.

Within a high priority period, if another station wants to start a multimedia session, it can use the bus simultaneously since it knows that the bus is being used for high priority traffic. However, the station has to send a (request) message to other stations. This message will be used to update the time at which the high priority period will terminate. In addition, stations will increment the number of current multimedia sessions by one. During this time period, high priority traffic compete for the bus using the CSMA/CD medium access protocol (but they do not compete with low priority traffic).

After a station has finished sending its high priority traffic, it broadcasts a completion message indicating that the number of multimedia sessions is decremented by one. Again the scheme has to ensure that the completion message gets delivered even after it collide with other traffic.

Thus the bus is time multiplexed between high priority and low priority traffic, as illustrated by Figure 1, where Tm is time period used by multimedia (high priority) traffic and Tr is time period during which regular (low priority) traffic can be sent.

The period (Tm) during which the bus time is dedicated to multimedia traffic is measured from the start of the high priority traffic session(s). This period ends when the count of multimedia sessions goes down to zero or if Tm exceeds a pre-determined maximum limit. If Tm exceeds this limit, regular traffic will be allowed to use the bus. The ratio Tm/Tr need to be chosen carefully to insure fairness to regular data traffic.

The number of simultaneous multimedia sessions has to be limited, otherwise, they would interfear with each other producing low quality in addition to increasing the regular data traffic delay. This limitation is dependent on the amount of available bandwidth and the nature of the multimedia applications.

This technique should result in better multimedia traffic performance over the non-prioritized bus protocol, but regular data traffic will suffer some delay. This delay could be acceptable to a user sending regular data. On the other hand, a delay in video frames, for example, may result in an unacceptable presentation quality.

4.2 Station Multiplexing Scheme

In the station multiplexing scheme, each station keeps status information about the transmission requests of all other stations. Using this status information, each station will be able to make decision to send packets on the bus. Each station is still required to send a request message before starting its multimedia session (i.e., high traffic session), but stations are not required to send a message after completing their sessions.

In this scheme, each station maintains a data structure that has an entry for each station. The entry for a station contains the time of the latest transmission request made by that station for high priority traffic. An entry for a station is updated when a (request) message is received indicating that it wants to start its multimedia session.

All stations use a fixed length quantum (Tq) for high priority traffic. A station uses the bus for this quantum and renews its request for another after the current quantum expires. A station wanting to start a session or renew its request for another quantum, compares the timestamps in its local structure. The station that corresponds to the smallest timestamp is the one that can use the bus next (after the current quantum expires).

A station can find if the current quantum has expired by comparing the difference between the latest (highest) timestamps in its local structure and the current time with the default session length quantum. If the current time minus the value of the latest timestamp is less than or equal the value of Tq, the station may proceed with its request immediately. Otherwise, the station tries whenever the current quantum expires.

In this scheme, the bus time is still multiplexed between high priority and low priority traffics, but in addition, the Tm period is further divided into small time quanta (of duration Tq) and each station uses a quantum (if needed) based on the latest timestamp criterion stated above. This would improve performance further since the competition between stations (during Tm) is regulated. In other words, this approach further reduces the possibility of collisions.

The basic idea is that stations that need to send high priority traffic will use the bus in a last recently used order. The station with the oldest (smallest) timestamp will be allowed to use the bus once the current Tq expires.

If two stations send their request messages at the same time, these messages will collide. In such cases, one station will backoff its transmission of request message and the other will proceed. (Station numbers can be used for tie breaking.)

As in the previous scheme, the period (Tm) during which the bus is dedicated to multimedia traffic has a maximum length from the start of the high priority traffic session(s). If Tm exceeds this limit, regular traffic will be allowed to use the bus. Again, the ratio Tm/Tr need to be chosen carefully to ensure fairness to regular data traffic.

A variation of this scheme is round robin scheme, where stations are served in a pre-defined order and for a fixed period of time. This scheme is similar to a token based scheme, where stations pass a token among themselves in a pre-defined order and only the station that has the token can use the bus for its high priority traffic for a predefined period. When timer expires, station passes the token to the next one. While this scheme reduces the competition between stations trying to send high priority traffic, it may result in an unacceptable response time for some demanding users.

5 Simulation Results

An event-driven, discrete time simulators of the proposed priority schemes was written to study their performance under various mixes of high priority and low priority traffics. The simulators were written in CSIM.

The performance measure studied was the mean response time or average end-to-end delay. It is the time interval between the instant a packet is ready to be transmitted and the instant the packet is delivered.

We studied the performance of the following three schemes:

- A no priority scheme: this schemes does not give priority to multimedia traffic.
- An absolute priority scheme: this schemes gives absolute priority to multimedia traffic. Low priority traffic had to wait whenever there is high priority traffic to be sent.
- Bus-time multiplexing scheme: this schemes divides the time into two intervals. One during which only multimedia traffic can be sent and another where both traffic types can be sent.

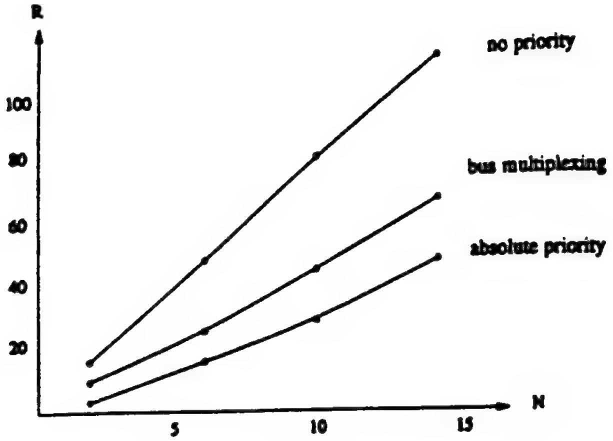


Figure 2. Response time vs. number of stations for 1K bits packets

The first scheme, namely the non priority scheme was studied to compare its performance with the other priority schemes. This scheme basically simulates an Ethernet/bus protocol. It is expected that the average end-to-end delay for this scheme to be high and not suitable for multimedia traffic.

The abolute priority works as follows. Each station gives higher priority to multimedia traffic. Any low priority packects will not be transmitted as long as there are some high priority traffic need to be transmitted. This scheme gives the best performance average response time (a lower bound on delay) for high priority traffic. On the other hand, it will result in the worst performance for low priority traffic.

The simulator used the parameters of a 10 Base 5 Ethernet that has a bandwidth of 10 Mbps. The maximum propagation delay is 2.5 micro seconds. A packet of size 1K bits have 100 micro seconds transmission time. Unless otherwise stated, the probability that the next request for the bus is for high priority traffic equals 0.5.

Figures 2 and 3 show the mean response time (in ms) for the three schemes as the number of workstations (multimedia sessions) increase. In Figure 2, packet size is 1K bits, while in Figure 3 the packet size is 8K bits.

We observe that the absolute priority scheme give the best results for high priority traffic. It improves the response time of high priority traffic by a about 60 percent. However, we found that the performance of low priority traffic performance was severely degraded. It degraded the performance of regular traffic by a factor of 35. In addition, it completely shut off regular traffic as the number of stations (multimedia sessions) increased beyond ten.

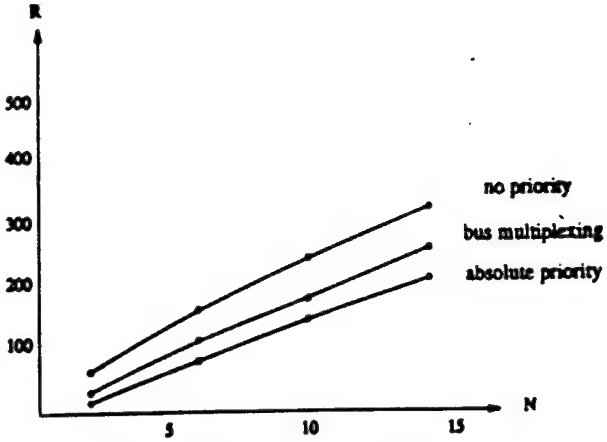


Figure 3. Response time vs. number of stations for 8K bits packets

The bus-time multiplexing scheme produced better overall results for both low priority and high priority traffic than a non-prioritized scheme or an absolute priority scheme. It improved the response time for high priority traffic by a about 35 percent, and degraded the performance of regular traffic by only a factor of 6.

Figure 4 shows the probability of discarding high priority packects due to their arrival after their deadlines for the three schemes under study. We assumed that packet size is 1K bits and a high priority packet misses its deadline (i.e., is of no value to the application) if it does not reach its destination in 100 ms.

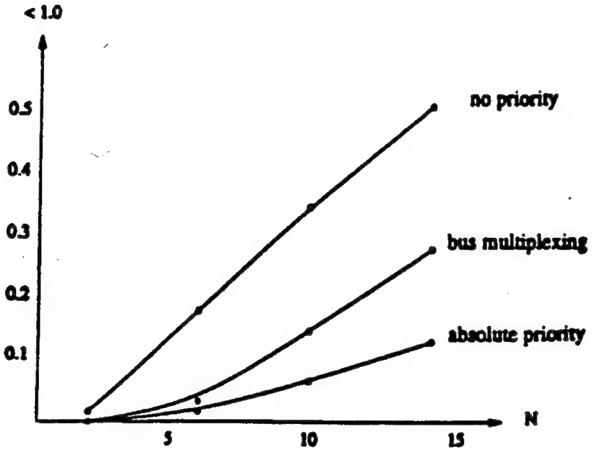


Figure 4. Probability of Discarding Packets

We observe that the absolute priority scheme reduces the probability of discarding high priority packet by a factor of 5. The bus-time multiplexing scheme reduces the probability of discarding high priority packet by a factor of 3. Note that this is quite an improvement over the no-priority scheme that has a relatively high deadline miss rate which adversely affects the quality of service.

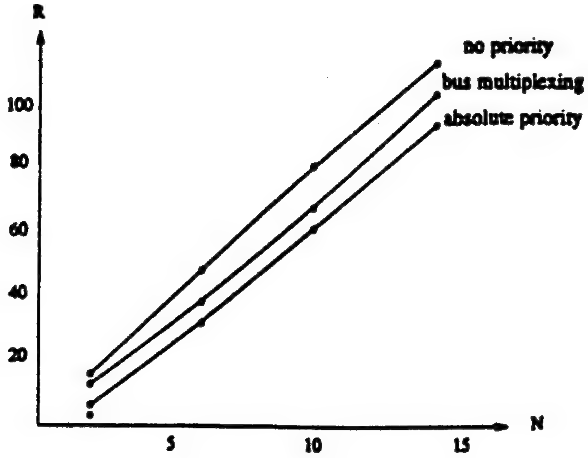


Figure 5. Response time for High Volume Multimedia Traffic

Figure 5 shows the results for a system where 80 percent of the transmission requests are for high priority traffic and 20 percent for regular traffic. By comparing Figure 2 and Figure 5 we observe that the performance of high priority traffic suffers when the fraction of high probability traffic is higher. This is due to the fact that there is more contention among high priority requests on the bus.

We observe that the absolute priority scheme still give the best performance for the high priority traffic. However, the bus-time multiplexing scheme gives the best overall performance for both types of traffics.

6 Conclusions

Traditional medium access protocols for Ethrenet do not support the concept of priority for data traffic. Stations along the bus equally compete for bus bandwidth. This means that packets can indiscriminately collide and get retransmitted. This medium access protocol is suitable for regular data traffic but not for multimedia traffic. The problem with multimedia streams is that they are isochronous in nature and they need to be delivered in real time. Thus a mixture of (multimedia and regular) traffic on a LAN bus would not satisfy this requirement. Multimedia traffic need to be transmitted at higher priority than regular data traffic.

This paper focused on supporting timely delivery of multimedia traffic on local area networks. We introduced two schemes, the bus-time multiplexing scheme and the station multiplexing scheme, for priority transmission of data on an Ethernet. These schemes add a sense of priority to the traffic on the bus.

The bus-time multiplexing scheme supports two levels of priority (high priority and low priority) and multiplexes the bus time between the two traffic types. It relies on sending a message to inform all stations that bus should be used for high priority traffic for some period of time during which low priority traffic are not transmitted. This technique results in a better multimedia traffic performance over the no-priority bus protocol but regular data traffic will suffer some delay. This delay could be acceptable to a user sending regular data. On the other hand, a delay in video frames, for example, may result in an unacceptable presentation quality.

In the station multiplexing scheme, the bus time is still multiplexed between high priority and low priority traffic, but in addition, the high-priority-period is further divided into small time quanta and each station uses a quantum (if needed) based on the last recently used criteria. This improves performance further since the competition between stations during the high-priority-period is regulated which in effect reduce the number of collisions.

The two schemes presented here reserve bus time for a specified time duration, while the scheme presented in [15] provides message-based priority functions. At the end of each transmission each station evaluates its priority compared to the priority of the current transmission. In our schemes, during the time reserved for high priority traffic, there is no need for request/reservation messages which in effect reduces the protocol overhead.

Simulation results indicate that the bus-time multiplexing scheme gives the best overall performance for both types of traffics compared with a non-prioritized scheme or a scheme that gives absolute priority to multimedia traffic.

REFERENCES

1. S. Ramanathan and Venkat Rangan, "Feedback Techniques for Intra-Media Continuity and Inter-Media Synchronization in Distributed Multimedia Systems," The Computer Journal, Vol. 36, No. 1, 1993, 19-31.

2. T. D. C. Little and A Ghafoor, "Multimedia Synchronization Protocols for Broadband Integrated Services," IEEE Journal on Selected Areas in Communication, 1991, 1368-1482.

3. R. B. Dannenberg, "Remote Access to Interactive Media," SPIE, Vol. 1785, Enabling Technologies for High-Bandwidth Applications, 1992, 230-237.

4. R. Steinmetz, "Synchronization Properties in Multimedia Systems," IEEE Journal on Selected Areas in Communication, 1990, 401-412.

5. C. Nicolaou, "An Architecture for Real-Time multimedia Communication System," IEEE Journal on Selected Areas in Communication, Vol. 8, No. 3, April 1990, 391-400.

6. Bernard Cole, "The Technology Framework," IEEE Spectrum, Vol 30, No. 3, March 1993, 32-39.

7. Amr Elsaadany, Mukesh Singhal and Ming T. Liu, "Multimedia on Local Area Networks," Technical Report OSU-CISRC-3/94-TR11, CIS Dept., OSU, March, 1994.

8. Jeff Burger, "The Desktop Multimedia Bible," Addison Wesley, 1993.

9. William Stallings, "Data and Computer Communications," Macmillan Publishing, Second Edition, 1988.

10. Andrew S. Tanenbaum, "Computer Networks," Printce Hall, Second Edition, 1989

11. Fouad A. Tobagi, "Multimedia: The Challenge Behind the Vision," Data Communications, Jan 21, 1993.

12. Daniel Minoli, "Isochronous Ethernet: Poised for Launch," Network Computing, August 1993, 156-162.

13. William Stallings, "ISDN and Broadband ISDN," Macmillan Publishing, Second Edition, 1992.

14. Khaled Amer, Ken Christensen and Tom Toher, "Experiments with Client/Server Multimedia on Token Ring," Proceedings of 18th Conference on Local Computer Networks, September 1993, 2-6.

15. Fouad A. Tobagi, "Carrier Sense Multiple Access With Message-Based Priority Functions," IEEE Trans. on Communications, Vol. COM-30, Jan. 1982, pp. 185-200.

Priority Communication Schemes on Local Area Networks for Multimedia Traffic

Amr Elsaadany, Mukesh Singhal and Ming T. Liu*
Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210

Abstract

In this paper we address the issues related to the delivery of multimedia streams on local area networks. Multimedia integrates voice and video along with text and images into existing systems. Traditionally, local area networks were designed to handle regular data traffic which are bursty in nature and for which variable delay is acceptable. Multimedia traffic, on the other hand, requires constant delay in addition to fast (or real time) delivery. Multimedia traffic also requires large bandwidth compared to regular traffic. Since local area networks are widely in use, their modification to integrate multimedia streams is very important. In this paper we propose priority-based communication schemes for the timely delivery of multimedia traffic on local area networks. We compare the performance of these schemes using simulation techniques.

Key words: Multimedia, local area networks, priority.

1 Introduction

Multimedia is a media that integrates the transmission of voice and video along with text and images into existing systems [1,2,3,4,5]. Digital multimedia transmits multimedia information digitally and allows them to be manipulated and stored on a computer system. It also allows for interactive human interface. Interactive multimedia allows the user to interface with the system in real time which is important for education, business, entertainment, and communications.

A large number of commercial organizations as well as universities use some type of local area network environment. These organizations have two options for adding multimedia capabilities. One option is to utilize the existing networks by adding some devices and/or software in order to transmit multimedia streams. The other option is to use an ATM type network (see section 3.2). ATM is designed such that voice and video can be transmitted effectively along with data packets. ATM switches and interfaces will be needed which would add a lot of cost to these organizations.

Existing local area networks have two main problems with regard to the integration of multimedia streams. First, the existing bandwidth capability for a standard LAN does not support multimedia transmission requirements. Second, there is no explicit priority mechanism for transmitting data/stream on LANs. Multimedia streams must be received at the user/presentation site at a time that there would be no interruption (or gap) from user's point of view (consider a moving picture presentation). Note that multimedia streams are isochronous and require a constant delay between packets.

This paper focuses on the problems associated with supporting multimedia on local network environments. The protocols of the local area networks are not suitable for multimedia traffic. They were designed to handle regular data traffic which are bursty in nature and for which variable delay is acceptable. Traditional communication protocols for local area networks have to be modified to suit the needs of multimedia applications. We present protocols for priority-based communications on Ethernets to allow timely delivery of multimedia traffic on Ethernets and compare the performance of the possible alternatives.

* Research reported herein was partially supported by U.S. Army Research Office, under contracts No. DAAL03-91-G-0093 and No. DAAL03-92-G-0184. The views, opinions, and/or findings contained in this paper are those of the authors and should not be construed as an official Department of the Army position, policy or decision.

0-8186-6680-3/94 $04.00 © 1994 IEEE

The rest of the paper is organized as follows: In the next section, we present data requirements of multimedia systems. In Section 3, we give a brief background of communication networks. Section 4 presents two techniques for priority-based communication on Ethernets. In Section 5, we present a simulation study of the proposed techniques. Finally, Section 6 concludes the paper.

2 Multimedia Data Requirements

All digital media requires digital end-to-end transmission of multimedia data streams. This means that the communication network must support the transmission of digitized voice, video and images. The network must also have a wide bandwidth to support these transmissions. Not all existing networks have sufficient bandwidth to accommodate the needs of all kinds of multimedia applications. For example, a LAN with 10 Mbps bandwidth may be able to provide limited support to transfer video and images for some multimedia applications.

Multimedia applications require fast processing and small visual response time. Moreover the effect of latency in transmitting streams between stations should be minimized or eliminated.

Moreover the amount of storage required by multimedia data is much more than that of regular data. A 640 x 480 24-bit color resolution graphic image, for example, takes about 900KB of memory. Most common audio CD uses 16-bit resolution and 44.1KHz sampling rate. One second of this high quality audio requires 176.4 kB of storage. Vedio is composed of a stream of frames (e.g., NTSC uses 30 per second). One second of 30 fps video of size 640 x 480 pixels and 24 bit color resolution requires about 27MBps bandwidth.

Multimedia data compression techniques reduces the amount of bandwidth required by streams such as video. In addition, the amount of disk storage required for multimedia data can be substantially decreased. The 27MBps transfer rate required for the 640 x 480 screen would be reduced to 550kBps using the MPEG1 standard [6]. MPEG1 can also compress audio by a factor of 5-10.

It is clear that the amount of data needed to represent video digitally places tremendous requirements on storage, transmission, bandwidth as well as display capabilities of current systems. In addition to the storage and bandwidth requirements for multimedia data, there are some other problems associated with multimedia such as the suitability of existing network protocols. More information about the properties of the multimedia data types/streams and the requirements of multimedia applications can be found in [7,8].

3 Communication Network Background

3.1 Local Area Networks

Local area networks (LAN) exist in one of the following architectures; bus, tree, token ring and FDDI (Fiber Dual Data Interface) ring [9,10]. The most widely used LAN is the Ethernet which is a bus-based architecture. This type of network typically has 10 Mbps bandwidth as opposed to 100 Mbps for FDDI.

Local area network bus (non token) protocols are based on broadcasting. The most dominating medium access protocol (MAC) in use is the CSMA/CD (Carrier Sense Multiple Access with Collision Detection) in which each station listens to the channel while transmitting. In case a collision is detected during transmission, the station ceases transmission and retries after some (random) period of time. This protocol requires that the message size be longer than twice the propagation time (in order to detect collision before transmission is complete). It also use binary exponential backoff technique, in which, after each repeated collision, the mean value of the random delay is doubled.

One other MAC that is widely used is the token bus in which stations on the bus form a logical ring with a token (control packet) to regulate access to the medium. When a station receives the token, it get control of the medium for a specified time. When the station is done, or time expires, it passes the token to the next station in the logical ring.

LAN topologies are based on sharing the bandwidth between stations. Data sent are assumed to be time-independent, they can be broken into packets without regard to their order. Also, when a station is using the network for transmission, no other station can and the entire bandwidth is assigned to it.

Options for Multimedia:

To overcome the problems with the speed and/or order of transmission on LAN, several alternatives are being explored. Fast Ethernet is one way to solve the bandwidth problem of the standard Ethernet and it is expected to be 10 times wider. It may retain the CSMA/CD MAC protocol to allow internetworking with standard Ethernet LANs.

Another way to overcome the bandwidth problem is to use switching hubs. Switched Ethernet establishes a single point-to-point LAN between each workstation and a high-capacity switching hub [11]. This gives each workstation its own LAN with all of its bandwidth. (Some topologies may have more than one workstation per LAN.) However the switching hub may be a bottleneck at some point. That is why this network architecture imposes limits on the number of workstations that could be connected to the hub, and also on the number of simultaneous users.

Other options include using FDDI (running over twisted pair for a short distance) to increase the bandwidth to 100 Mbps. FDDI technology could be extended to support the timely delivery required by multimedia applications. Isochronous Ethernet is another option which provides isochronous channels to stations equipped with isoENET cards [12].

3.2 Switching Networks

Switching networks transfer information between source and destination via three functions: transmission, multiplexing, and switching. Transmission provides the point-to-point transfer of information. Multiplexing uses a single transmission facility (link) for several independent channels. Switching uses properties of telecommunications traffic to replace a fully meshed network interconnecting all pairs of users with a shared substructure providing on demand connections, that is, packets can be assinged to channels based on the address information in the header of the packets themselves.

Circuit switching networks use time division multiplexing (TDM) where bandwidth is multiplexed between several channels with assigned amounts of bandwidth. Time slots within a frame are allocated to the channel for the duration of the service. This type of networks is suitable for applications that require guaranteed bandwidth or low latency such as PCM voice and constant bit rate data traffic. The mode of transfer in this type of network is referred to as synchronous transfer mode (STM).

Packet switching networks use bandwidth efficiently to suit bursty applications such as file transfers. However, the delay in these networks is unpredictable and the latency tends to be high. Packet switching uses statistical multiplexing which dynamically allocates time slots to incoming channels based on demand. In contrast with synchronous TDM, the positional significance of the slots (in TDM) is lost and hence each packet has to carry addressing information in addition to data.

ATM (Asynchronous Transfer Mode) uses cell switching which uses bandwidth flexibly and efficiently for all sorts of applications. For constant rate applications, a guaranteed number of cells per second can be allocated and any unused cell slots can be allocated (on demand) to bursty applications. In addition, the number of virtual channels carrying these cells can vary as well as the rate of each channel.

In cell switching technology, a cell is a 53 byte fixed length packet with 5 bytes for header (containing routing and quality of service information) and 48 bytes for data (payload section) [13]. This fixed length cells are better suited for high speed switching because it enables simpler implementation of switches and provides predictable system behavior.

The asynchronous nature of ATM comes from the fact that cells are assigned asynchronously to any service based on its needs. This is facilitated by the fact that each cell is addressed independently to any destination. In this mode of transmission, there is no notion of owning a time slot on periodic basis.

Although existing wide networks (WAN) can be used by some multimedia applications, these WAN-based applications would suffer from existing problems such as the varying network delays. When it become widely available, ATM will be a suitable technology that has the high bandwidth and low latency requirements for many multimedia applications.

4 Priority Schemes

As mentioned before, our work focuses on the problems associated with supporting multimedia services on local network environments. The protocols for regular data transfer on the local area networks are not suitable for multimedia traffic and they have to be modified to suit multimedia applications. These protocols were designed to handle regular data traffic which is bursty in nature and for which variable delay is acceptable. We present priority-based communication protocols for allowing timely delivery of multimedia traffic on local area networks. We focus our attention to the problems associated with supporting multimedia traffic on with local area bus (Ethernet) as opposed to token ring networks [14].

The current medium access bus protocol does not provide priority for transfer of data. Stations on the bus use the CSMA/CD protocol to compete for bus bandwidth. This means packets can indiscriminately collide and get retransmitted. This medium access protocol is suitable for regular data traffic but not for multimedia traffic. We propose two priority schemes that add some sense of priority to data traffic on the bus. They support two levels of priority: high priority (for multimedia or real-time traffic) and low priority (for regular data).

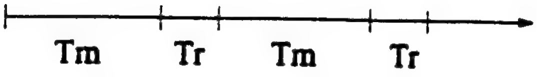


Figure 1. Bus Time

4.1 Bus-Time Multiplexing Priority Scheme

Bus-time multiplexing scheme relies on message passing to support two levels of priority. Before a station sends a "high priority" traffic (for a multimedia session), it informs every other station of its intention so that they would abstain from sending "low priority" traffic for a period of time. This is accomplished by broadcasting a request message that includes the amount of time that the station needs (i.e., the session length). During this time period, only high priority traffic is allowed to use the bus.

Since the request message will compete for the bus with other traffic, the scheme has to ensure that this message gets successfully transmitted before the high priority traffic period starts.

When stations receive a message indicating that high priority traffic need to be sent by a station, they stop sending low priority traffic at once. Stations record the time at which they will be allowed to send regular traffic. This time would be equivalent to the current time plus the length of the session (which is included in the request message). Stations also keep a count of the number of multimedia sessions currently in progress. Whenever a station receives a request message, it increment this count by one.

Within a high priority period, if another station wants to start a multimedia session, it can use the bus simultaneously since it knows that the bus is being used for high priority traffic. However, the station has to send a (request) message to other stations. This message will be used to update the time at which the high priority period will terminate. In addition, stations will increment the number of current multimedia sessions by one. During this time period, high priority traffic compete for the bus using the CSMA/CD medium access protocol (but they do not compete with low priority traffic).

After a station has finished sending its high priority traffic, it broadcasts a completion message indicating that the number of multimedia sessions is decremented by one. Again the scheme has to ensure that the completion message gets delivered even after it collide with other traffic.

Thus the bus is time multiplexed between high priority and low priority traffic, as illustrated by Figure 1, where Tm is time period used by multimedia (high priority) traffic and Tr is time period during which regular (low priority) traffic can be sent.

The period (Tm) during which the bus time is dedicated to multimedia traffic is measured from the start of the high priority traffic session(s). This period ends when the count of multimedia sessions goes down to zero or if Tm exceeds a pre-determined maximum limit. If Tm exceeds this limit, regular traffic will be allowed to use the bus. The ratio Tm/Tr need to be chosen carefully to insure fairness to regular data traffic.

The number of simultaneous multimedia sessions has to be limited, otherwise, they would interfear with each other producing low quality in addition to increasing the regular data traffic delay. This limitation is dependent on the amount of available bandwidth and the nature of the multimedia applications.

This technique should result in better multimedia traffic performance over the non-prioritized bus protocol, but regular data traffic will suffer some delay. This delay could be acceptable to a user sending regular data. On the other hand, a delay in video frames, for example, may result in an unacceptable presentation quality.

4.2 Station Multiplexing Scheme

In the station multiplexing scheme, each station keeps status information about the transmission requests of all other stations. Using this status information, each station will be able to make decision to send packets on the bus. Each station is still required to send a request message before starting its multimedia session (i.e., high traffic session), but stations are not required to send a message after completing their sessions.

In this scheme, each station maintains a data structure that has an entry for each station. The entry for a station contains the time of the latest transmission request made by that station for high priority traffic. An entry for a station is updated when a (request) message is received indicating that it wants to start its multimedia session.

All stations use a fixed length quantum (Tq) for high priority traffic. A station uses the bus for this quantum and renews its request for another after the current quantum expires. A station wanting to start a session or renew its request for another quantum, compares the timestamps in its local structure. The station that corresponds to the smallest timestamp is the one that can use the bus next (after the current quantum expires).

A station can find if the current quantum has expired by comparing the difference between the latest (highest) timestamps in its local structure and the current time with the default session length quantum. If the current time minus the value of the latest timestamp is less than or equal the value of Tq, the station may proceed with its request immediately. Otherwise, the station tries whenever the current quantum expires.

In this scheme, the bus time is still multiplexed between high priority and low priority traffics, but in addition, the Tm period is further divided into small time quanta (of duration Tq) and each station uses a quantum (if needed) based on the latest timestamp criterion stated above. This would improve performance further since the competition between stations (during Tm) is regulated. In other words, this approach further reduces the possibility of collisions.

The basic idea is that stations that need to send high priority traffic will use the bus in a last recently used order. The station with the oldest (smallest) timestamp will be allowed to use the bus once the current Tq expires.

If two stations send their request messages at the same time, these messages will collide. In such cases, one station will backoff its transmission of request message and the other will proceed. (Station numbers can be used for tie breaking.)

As in the previous scheme, the period (Tm) during which the bus is dedicated to multimedia traffic has a maximum length from the start of the high priority traffic session(s). If Tm exceeds this limit, regular traffic will be allowed to use the bus. Again, the ratio Tm/Tr need to be chosen carefully to ensure fairness to regular data traffic.

A variation of this scheme is round robin scheme, where stations are served in a pre-defined order and for a fixed period of time. This scheme is similar to a token based scheme, where stations pass a token among themselves in a pre-defined order and only the station that has the token can use the bus for its high priority traffic for a predefined period. When timer expires, station passes the token to the next one. While this scheme reduces the competition between stations trying to send high priority traffic, it may result in an unacceptable response time for some demanding users.

5 Simulation Results

An event-driven, discrete time simulators of the proposed priority schemes was written to study their performance under various mixes of high priority and low priority traffics. The simulators were written in CSIM.

The performance measure studied was the mean response time or average end-to-end delay. It is the time interval between the instant a packet is ready to be transmitted and the instant the packet is delivered.

We studied the performance of the following three schemes:

- A no priority scheme: this schemes does not give priority to multimedia traffic.
- An absolute priority scheme: this schemes gives absolute priority to multimedia traffic. Low priority traffic had to wait whenever there is high priority traffic to be sent.
- Bus-time multiplexing scheme: this schemes divides the time into two intervals. One during which only multimedia traffic can be sent and another where both traffic types can be sent.

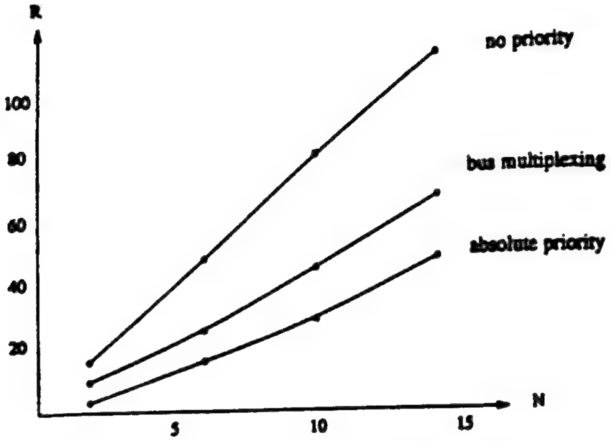


Figure 2. Response time vs. number of stations for 1K bits packets

The first scheme, namely the non priority scheme was studied to compare its performance with the other priority schemes. This scheme basically simulates an Ethernet/bus protocol. It is expected that the average end-to-end delay for this scheme to be high and not suitable for multimedia traffic.

The abolute priority works as follows. Each station gives higher priority to multimedia traffic. Any low priority packects will not be transmitted as long as there are some high priority traffic need to be transmitted. This scheme gives the best performance average response time (a lower bound on delay) for high priority traffic. On the other hand, it will result in the worst performance for low priority traffic.

The simulator used the parameters of a 10 Base 5 Ethernet that has a bandwidth of 10 Mbps. The maximum propagation delay is 2.5 micro seconds. A packet of size 1K bits have 100 micro seconds transmission time. Unless otherwise stated, the probability that the next request for the bus is for high priority traffic equals 0.5.

Figures 2 and 3 show the mean response time (in ms) for the three schemes as the number of workstations (multimedia sessions) increase. In Figure 2, packet size is 1K bits, while in Figure 3 the packet size is 8K bits.

We observe that the absolute priority scheme give the best results for high priority traffic. It improves the response time of high priority traffic by a about 60 percent. However, we found that the performance of low priority traffic performance was severely degraded. It degraded the performance of regular traffic by a factor of 35. In addition, it completely shut off regular traffic as the number of stations (multimedia sessions) increased beyond ten.

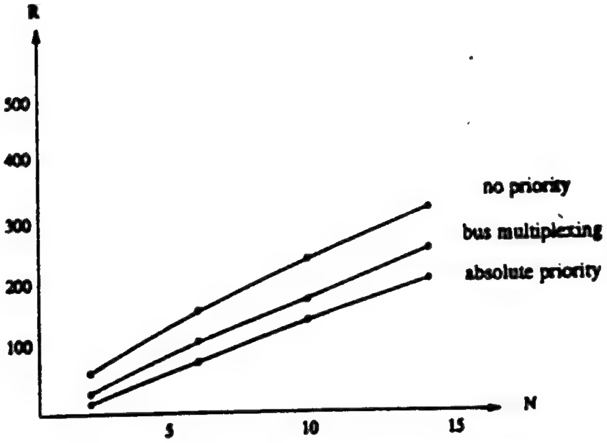


Figure 3. Response time vs. number of stations for 8K bits packets

The bus-time multiplexing scheme produced better overall results for both low priority and high priority traffic than a non-prioritized scheme or an absolute priority scheme. It improved the response time for high priority traffic by a about 35 percent, and degraded the performance of regular traffic by only a factor of 6.

Figure 4 shows the probability of discarding high priority packects due to their arrival after their deadlines for the three schemes under study. We assumed that packet size is 1K bits and a high priority packet misses its deadline (i.e., is of no value to the application) if it does not reach its destination in 100 ms.

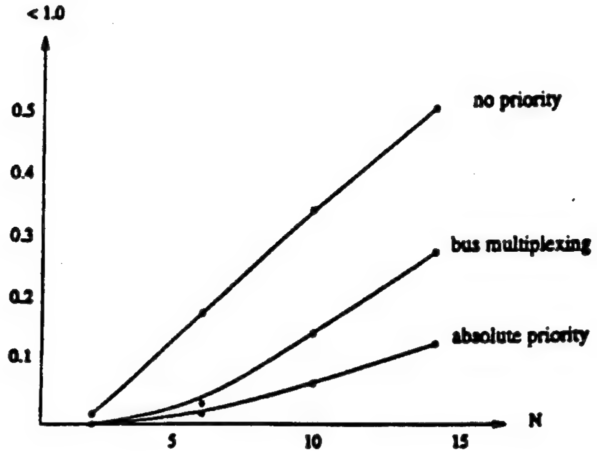


Figure 4. Probability of Discarding Packets

We observe that the absolute priority scheme reduces the probability of discarding high priority packet by a factor of 5. The bus-time multiplexing scheme reduces the probability of discarding high priority packet by a factor of 3. Note that this is quite an improvement over the no-priority scheme that has a relatively high deadline miss rate which adversely affects the quality of service.

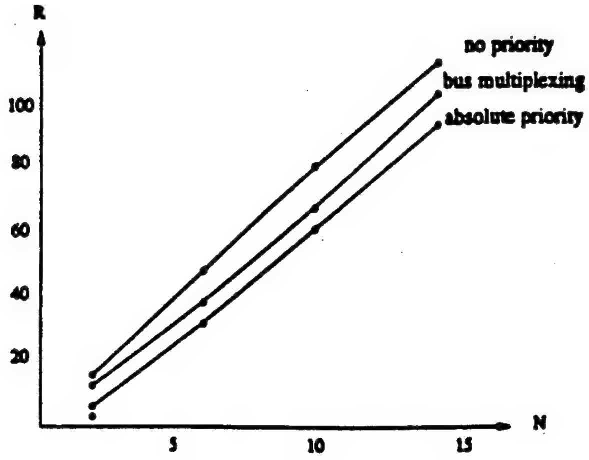


Figure 5. Response time for High Volume Multimedia Traffic

Figure 5 shows the results for a system where 80 percent of the transmission requests are for high priority traffic and 20 percent for regular traffic. By comparing Figure 2 and Figure 5 we observe that the performance of high priority traffic suffers when the fraction of high probability traffic is higher. This is due to the fact that there is more contention among high priority requests on the bus.

We observe that the absolute priority scheme still give the best performance for the high priority traffic. However, the bus-time multiplexing scheme gives the best overall performance for both types of traffics.

6 Conclusions

Traditional medium access protocols for Ethrenet do not support the concept of priority for data traffic. Stations along the bus equally compete for bus bandwidth. This means that packets can indiscriminately collide and get retransmitted. This medium access protocol is suitable for regular data traffic but not for multimedia traffic. The problem with multimedia streams is that they are isochronous in nature and they need to be delivered in real time. Thus a mixture of (multimedia and regular) traffic on a LAN bus would not satisfy this requirement. Multimedia traffic need to be transmitted at higher priority than regular data traffic.

This paper focused on supporting timely delivery of multimedia traffic on local area networks. We introduced two schemes, the bus-time multiplexing scheme and the station multiplexing scheme, for priority transmission of data on an Ethernet. These schemes add a sense of priority to the traffic on the bus.

The bus-time multiplexing scheme supports two levels of priority (high priority and low priority) and multiplexes the bus time between the two traffic types. It relies on sending a message to inform all stations that bus should be used for high priority traffic for some period of time during which low priority traffic are not transmitted. This technique results in a better multimedia traffic performance over the no-priority bus protocol but regular data traffic will suffer some delay. This delay could be acceptable to a user sending regular data. On the other hand, a delay in video frames, for example, may result in an unacceptable presentation quality.

In the station multiplexing scheme, the bus time is still multiplexed between high priority and low priority traffic, but in addition, the high-priority-period is further divided into small time quanta and each station uses a quantum (if needed) based on the last recently used criteria. This improves performance further since the competition between stations during the high-priority-period is regulated which in effect reduce the number of collisions.

The two schemes presented here reserve bus time for a specified time duration, while the scheme presented in [15] provides message-based priority functions. At the end of each transmission each station evaluates its priority compared to the priority of the current transmission. In our schemes, during the time reserved for high priority traffic, there is no need for request/reservation messages which in effect reduces the protocol overhead.

Simulation results indicate that the bus-time multiplexing scheme gives the best overall performance for both types of traffics compared with a non-prioritized scheme or a scheme that gives absolute priority to multimedia traffic.

REFERENCES

1. S. Ramanathan and Venkat Rangan, "Feedback Techniques for Intra-Media Continuity and Inter-Media Synchronization in Distributed Multimedia Systems," The Computer Journal, Vol. 36, No. 1, 1993, 19-31.

2. T. D. C. Little and A Ghafoor, "Multimedia Synchronization Protocols for Broadband Integrated Services," IEEE Journal on Selected Areas in Communication, 1991, 1368-1482.

3. R. B. Dannenberg, "Remote Access to Interactive Media," SPIE, Vol. 1785, Enabling Technologies for High-Bandwidth Applications, 1992, 230-237.

4. R. Steinmetz, "Synchronization Properties in Multimedia Systems," IEEE Journal on Selected Areas in Communication, 1990, 401-412.

5. C. Nicolaou, "An Architecture for Real-Time multimedia Communication System," IEEE Journal on Selected Areas in Communication, Vol. 8, No. 3, April 1990, 391-400.

6. Bernard Cole, "The Technology Framework," IEEE Spectrum, Vol 30, No. 3, March 1993, 32-39.

7. Amr Elsaadany, Mukesh Singhal and Ming T. Liu, "Multimedia on Local Area Networks," Technical Report OSU-CISRC-3/94-TR11, CIS Dept., OSU, March, 1994.

8. Jeff Burger, "The Desktop Multimedia Bible," Addison Wesley, 1993.

9. William Stallings, "Data and Computer Communications," Macmillan Publishing, Second Edition, 1988.

10. Andrew S. Tanenbaum, "Computer Networks," Printce Hall, Second Edition, 1989

11. Fouad A. Tobagi, "Multimedia: The Challenge Behind the Vision," Data Communications, Jan 21, 1993.

12. Daniel Minoli, "Isochronous Ethernet: Poised for Launch," Network Computing, August 1993, 156-162.

13. William Stallings, "ISDN and Broadband ISDN," Macmillan Publishing, Second Edition, 1992.

14. Khaled Amer, Ken Christensen and Tom Toher, "Experiments with Client/Server Multimedia on Token Ring," Proceedings of 18th Conference on Local Computer Networks, September 1993, 2-6.

15. Fouad A. Tobagi, "Carrier Sense Multiple Access With Message-Based Priority Functions," IEEE Trans. on Communications, Vol. COM-30, Jan. 1982, pp. 185-200.

D. **M. T. Liu, H.-W. Jeng, and L. S. Koh,**
"Formal Description Techniques for Protocol Specification,"
Proc. of ATR Int'l Workshop on Communications Software Engineering,
pp. 31–71, October 1994.

Formal Description Techniques
for
Protocol Specification*

Ming T. Liu, Hou-Wa J. Jeng, and Liang Seng Koh
Department of Computer and Information Science
The Ohio State University
2015 Neil Avenue
Columbus, OH 43210-1277

Abstract

Formal methods for protocol design have emerged since the last decade due to the recognition of the limitations of informal techniques by many researchers. Experience has shown that informal techniques that use ad hoc approaches, often produces systems with errors and undesirable behaviors. On the other hand, by using formal approaches, many phases of protocol design can be automated, thereby minimizing the occurrences of errors and undesirable behaviors. Formal description techniques (FDTs), which give a precise definition of the specified protocol and, consequently, allow for formal methods to examine the protocol correctness and performance, have become the basis of formal methods for protocol-design. In this paper, besides the standardized FDTs, such as LOTOS, Estelle and SDL, several other FDTs in various models are presented and compared. For ease of comparison, a common example, the alternate bit protocol (ABP), is used throughout this paper to demonstrate how various FDTs can be used in protocol specifications.

[1] Research reported herein was supported by U.S. Army Research Office, under contracts No.DAAL03-91-G-0093 and No. DAAL03-92-G-0184. The views, opinions, and/or findings contained in this paper are those of the authors and should not be construed as an official Department of the Army position, policy or decision.

1 Introduction

A protocol is a set of rules governing the exchange of messages between entities in a computer communication network. At the lowest level, protocols may prescribe how information is to be transmitted and received over a physical medium, and how that information is to be physically represented on the medium. At higher levels, protocols aim to overcome inherent unreliability in low levels, to prevent congestion and deadlocks, to control the flow of information, and to provide mechanisms for delivery, addressing and routing of messages. At still higher levels, protocols may provide services for transferring files between physically separated computers, for enabling communication between incompatible terminals, for ensuring security in data transmission, etc. Consequently, in computer networks, protocols are key components and generally complex. They must work correctly for the system to provide expected services.

Moreover, since protocols define the interaction between communicating entities running in parallel and residing at different nodes of the network, their design can be remarkably hard, much harder than it is to write a sequential program. Unfortunately, when the design of a new protocol is complete, the designers usually have little trouble convincing themselves that it is trivially correct. As a result, the designers usually decides to trust their instincts and foregoes the formal proofs. Subtle logical flaws in a design thus get a chance to hide, and inevitably find the worst possible moment in the lifetime of the protocol to reveal themselves.

Because of the complexity of protocol software, experience has shown that the use of informal techniques in protocol development often produces systems with errors and undesirable behaviors; thus, it has been recognized that *good* software engineering techniques are needed for the implementation and maintenance of these protocol software and systems. Consequently, formal methods for protocol design have emerged since the last decade. Using formal approaches, a protocol is represented by a formal model (or interchangeably, a formal specification). Analytic methods can then be used to examine logical correctness and performance of the protocol before it is actually implemented. This methodology has been proved to be so effective in identifying many protocol design errors that the discipline in this area is called *protocol engineering* [1].

Protocol engineering includes many phases: protocol specification, protocol validation, conformance testing, protocol conversion, etc. Among them, *protocol specification* is the basis, around which all other phases of protocol engineering evolve. In protocol engineering, FDTs (formal description techniques) are used for protocol specification such that subsequent formal protocol analysis and verification can be performed. So far, many FDTs have been proposed and examined by many researchers and different opinions on the strength and weakness of various FDTs exist. In [2], it is pointed out that, in general, an FDT that is suitable for protocol specification must be able to provide a basis for:

- the development of unambiguous, clear and concise specifications.
- the verification of specifications.
- the functional analysis of specifications.
- the development of implementations from specifications.
- the determination that an implementation conforms to its specification, i.e. conformance testing.

Furthermore, in [3], it is pointed out that a *good* FDT should contain features as follows:

- broad functional coverage such as concurrent aspects, data descriptions, etc.
- formal definition such as formally described syntax and interpretation models, etc.
- good human orientation such as readability, etc.
- high expressive power.
- structuring capability and good reusability.
- supporting tools such as graphical input interface, etc.

In fact, since the widespread acceptance of the OSI reference model and its standardized protocols, a great deal of research efforts and activities in protocol engineering and FDTs have been made to develop standardized FDTs. The aforementioned six features have been the targets for most of them.

The FDTs, which have been proposed and evaluated by various researchers and organizations, can be categorized according to the specification models they use as follows [1, 4]:

- state transition models - a communication protocol basically consists of event-driven entities that communicate with each other through message passing. The behavior of an entity can be described in terms of the actions (or state transitions) that the entity takes in response to external and internal events. State transition models provide the simplest models for describing protocol entities. The advantages of these models are that they are simple and many general properties of a protocol can be verified. However, they are not suitable for describing complex protocols since they involve too many states and transitions. Models falling into this category include finite-state automata, formal grammars, and Petri nets and their derivatives.

- programming models - a communication protocol executes an algorithmic procedure which can be described using a language notation such as a high level programming language. In general, a protocol description in these models is difficult to analyze for correctness. Efforts to prove the correctness of the program (the *safety* and *liveness* properties) far exceed those required for developing the program, and its correctness proof usually depends heavily on human ingenuity and is hard to automate.

- hybrid models - these models combine the features of the transition models and programming language models for describing protocols. In these models, state transition systems are augmented with variables and program fragments to give them added power. An example is the addition of such features as variables and program fragments to finite state machines, and they are called *extended finite state machines* (EFSM). A model of this type can capture the main feature of a protocol using a small number of states, while the variables and program fragments are used in capturing the program-like aspects of the protocol.

Each category has its inherent advantages and disadvantages. In this paper, several FDTs in all three categories, are briefly described and compared.

The rest of this paper is organized as follows: Section 2, Section 3 and Section 4 describe various FDTs in state transition models, programming language models and hybrid models, respectively; in Section 5, conclusions are presented.

2 State-Transition Models

2.1 Finite-State Automata (FSA)

The FSA model is one of the earliest formal models applied to protocols. Ever since Lynch (in 1968) [5] and subsequently Bartlett et al. (in 1969) [6] used FSA for specifying the Alternating Bit Protocol (ABP), the number of formal models for protocol specification has increased at a rapid rate. The ABP has since then become a classical example and been used extensively by other models to illustrate their feasibility in protocol specification and verification. Throughout this paper, ABP will be used as an example to show how each FDT can be used.

FSA models are based on the observation that protocols consist largely of relatively simple processing activities in response to a number of events such as commands from the user, message arrivals from another entity, and internal timeouts. Therefore, finite-state automata with such events forming their transitions are a natural model for specifying communication protocols. The basic approach is to specify the communication system as a collection of finite-state automata, each describing the behavior of a communicating entity.

In this model, a protocol is represented by a network of communicating finite-state automata and channels between protocol entities are modeled by FIFO queues. Each state of a finite-state automaton corresponds to a different control stage of the entity it represents. Each transition of a automaton is labeled with either an input event that enables the transition or an output event that takes place as part of the transition.

In Fig. 1, an example is shown how FSA is used to specify the Alternating Bit Protocol (ABP). This protocol provides reliable transmission of data from one communicating entity (called the sender) to the other (called the receiver). In Fig. 1, entity P_a is the sender and entity P_b is the receiver. Data are divided into frames, and frames are transmitted one at a time. Note that the channels between these two entities, which are not shown in Fig. 1, are assumed to be FIFO queues. The characteristics of the channels are: they can be noisy and data can be garbled but never lost. Garbled data will be detected and recovered by the protocol. The error detection is done by checking the sequence control bit of the header of each frame. Under normal transmission (without error), the sequence control bit of each frame header should alternate between 0 and 1 for successive frames. If the receiver detects an out of sequence frame, it sends a negative acknowledgment to ask for re-transmission. Otherwise, each correctly received frame is explicitly acknowledged. The negative acknowledgment is signaled by repeating the acknowledgment of the last acknowledged frame. Each data frame is retransmitted until a positive acknowledgment is received. Thus, in Fig. 1, P_a sends data frames D0 and D1 alternately to P_b. For each data frame sent by P_a, an acknowledgment, A0 for D0 and A1 for D1 respectively, is sent by P_b. The protocol user at P_a executes IN to

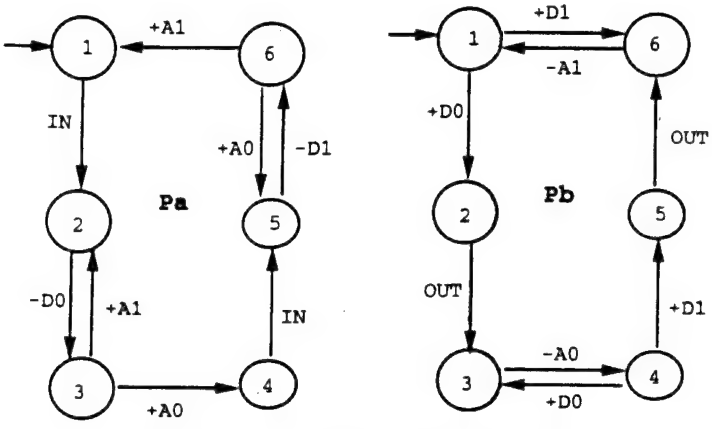


Figure 1: ABP in FSA Model

request sending of a data frame and when a data frame arrives at the other end, P_b executes OUT to pass the data frame to the user at P_b. Note that "+" denotes the reception of a data frame or acknowledgment and "-" denotes the sending of a data frame or acknowledgment.

Formally, a protocol P in this model is defined as a set of communicating protocol entities and a protocol entity is defined as follows.

A Protocol Entity P_x is a five-tuple $(q, \lambda, \delta, i, f)$, where:

1. q is the set of states in P_x.
2. λ is the finite set of transitions in P_x.
3. δ is the transition function of P_x which maps $q \times \cup\lambda$ into q; if $\delta(s_1, t) = s_2$, where $s_1 \in q, s_2 \in q$, and $t \in \lambda$, s_1 is called the starting state of t and s_2 is the ending state of t. Note that a *receive/read* transition t_r is said to be executable, only when the current state of P_x is the starting state of t_r and the *message to be received/read* is already in the incoming channel to P_x.
4. i is the initial state of P_x.
5. f is the set of the final states in P_x. Note that for a typical non-terminating protocol, f usually has only one element, i, which is the initial state of P_x.

Bochmann [7] used an FSA model to analyze the ABP and the X.25 call setup and clearing procedures. West and Zafiropulo [8] used an automated technique to analyze the X.21 and found a number of unspecified reception errors in the 1976 version, which were subsequently corrected in the 1980 version. Gouda and his associates [9, 10, 11] used a network of communicating FSA to model, analyze and synthesize protocols. Lee and Lai [12] have used a relational-algebra approach to represent an FSA as a transition table. On this basis, the well-known theory in relational database can be used to derive the global state transitions of the system. Furthermore, the logical errors of protocols can be formulated in terms of relational algebra. This approach have been implemented on the INGRES database system and applied to the validation of several protocols including the X.21.

A limitation of the FSA model is that all necessary information must be represented by explicit states. For example, there must be states and events to handle each possible sequence number. For complex protocols, the number of states required can be very large, thereby creating the so-called *state explosion problem*.

2.2 Formal Grammars

Formal languages and the grammars that define them are a type of the state-transition model. If one views the sequence of input and output of an FSA as sequences of a formal language, one can define the formal grammar that would produce all valid sequences. There is a well-known correspondence between such grammars and various types of automata that will recognize (or generate) all valid sequences of the language.

Harangozo [13] used regular grammars to specify the HDLC protocol and extended the model to handle sequence numbers by indexing the production rules of the grammar. Using context-free grammars, Teng and Liu [14, 15, 16] developed a Transition Grammar (TG) model for the design and implementation of communication protocols.

In the TG grammar, a protocol is represented by a set of formal grammars. As formal grammars are capable of defining a language, the idea is to come up with a set of production rules that define all the legal protocol action sequences. Each entity or channel of the protocol in the TG model is described by a regular grammar. Production rules in the grammar have the following form:

<left-non-terminal> ::= terminal-string <right-non-terminal>

Terminal symbols in the TG production rules represent protocol actions, and non-terminal symbols are equivalent to the states in the FSA model. The meaning of a production rule is that the entity in the state specified by the left-hand non-terminal may take the action specified by the terminal string and enter the state specified by the right-hand non-terminal.

Terminal actions in the TG model are the following: D (Dequeue), Q (enQueue), F (Fetch), P (Push), O (pOp), C (Clear), N (Non-empty), or U (fUll). Interested readers should refer to [16] for details. These actions not only enable modeling of a communication medium as FIFO, non-FIFO, and priority queues, but also make status checking of an output queue available. Consequently, they provide a model more powerful than the FSA model, while keeping the model still feasible for automatic verification. As an example, a TG specification of the ABP is shown as follows:

```
Sender Entity
-------------

<1> ::= IN              <2>,
<2> ::= Q.2.D0          <3>,
<3> ::= D.1.A0          <4>,
        D.1.A1          <2>,
<4> ::= IN              <5>,
<5> ::= Q.2.D1          <6>,
<6> ::= D.1.A1          <1>,
        D.1.A0          <5>,


Receiver Entity
---------------

<1> ::= D.2.D0          <2>,
        D.2.D1          <6>,
<2> ::= OUT             <3>,
<3> ::= Q.1.A0          <4>,
<4> ::= D.2.D1          <5>,
        D.2.D0          <3>,
<5> ::= OUT             <6>,
<6> ::= Q.1.A1          <1>,
```

Note that in the example, queue 2 (1) is the FIFO queue of the channel from the sender (receiver) entity to the receiver (sender) entity. For example, Q.2.D0 is the action of putting data frame D0 into the head of queue 2 (from sender entity to receiver entity); likewise, D.1.A0 is the action of dequeuing the head message from queue 1 (from receiver entity to sender entity) and examining if the message is acknowledgment A0.

The TG model has been automated [17] and used to validate the X.21 and the call setup procedure of the TCP [18, 19, 20]. It has also been extended to handle timing constraints (called the TTG model [17]).

2.3 Petri-Nets

Petri Nets belong to the class of state transition models, but have a theoretical and practical applicability broader than that of finite state machines. Diaz [21] made an extensive survey on how Petri nets are used to specify and analyze protocols. In the Petri nets model, a protocol is modeled by a number of component nets representing different protocol entities. Basically, a Petri net is a graph containing a set of *places* (represented by circles) and a set of *transitions* (represented by bars). Directed arcs are used to connect places to transitions, and transitions to places. A number of token distributed in the places represent a marking of the net and also decide which transitions are firable. The firing of a transition causes a redistribution of tokens, and thus moves the net to a new marking. Therefore, places and transitions of a Petri net specify conditions and events, respectively. How places and transitions are connected can be used to describe the behavior of a protocol.

Formally, a protocol P in the Petri nets model is defined as a quadruple (P,T,E,m_0), where

1. P is a finite nonempty set of places.
2. T is a finite nonempty set of transitions.

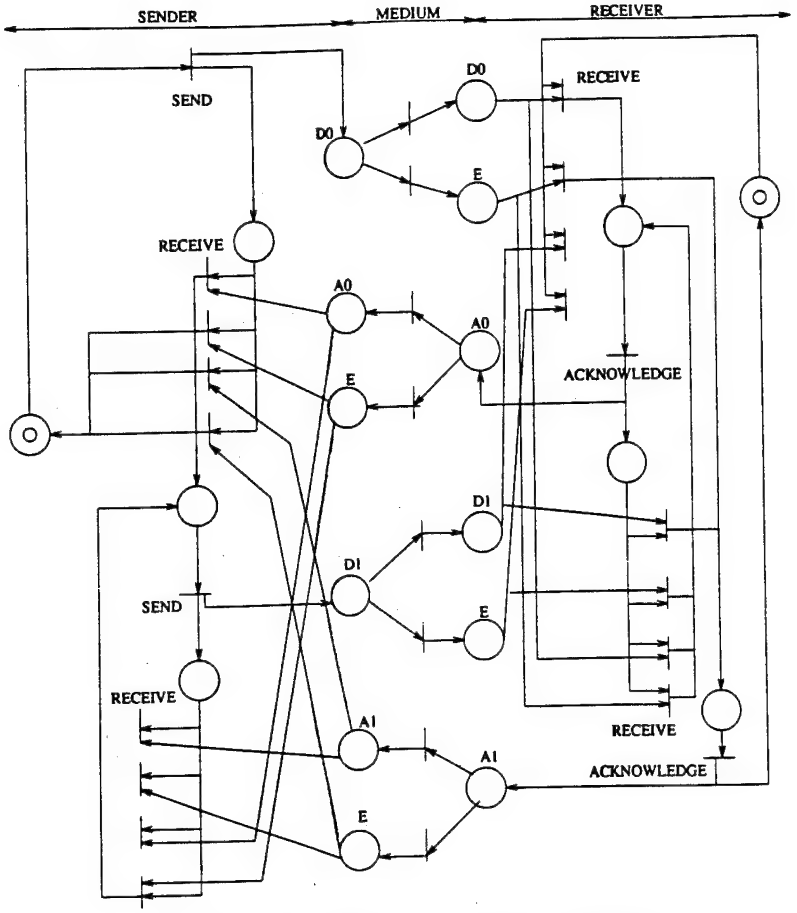


Figure 2: ABP in Petri Nets Models

3. E is a set of directed arcs, $E \subseteq P \times T \cup T \times P$, such that for each $t \in T$,

$$(P_i, t) \in E \wedge (t, P_j) \in E \Rightarrow (P_i, P_j \in P).$$

4. m_0 is an initial marking function that assigns a nonnegative integer number of tokens to each place of the net.

A transition is defined to be firable by a marking m iff every input place of this transition contains at least one token. When a transition is fired, a token is removed from each of its input places and a token is added to each of its output places. This leads the net to a new marking. Fig. 2 shows how the ABP is specified in the Petri nets model.

Petri nets have been used to specify and verify ABP [22, 23] and the call-setup procedure of a packet-switched network [24], and the ISO Transport protocol [25]. A variation of the Petri net model, called SARA, has been used to model the X.21 [26] and the X.25 [27]. In [28], timing constraints are added to Petri nets (called timed Petri nets). With timing information available in Petri nets, Petri nets have also been used to evaluate the performance of the system they modeled [29, 30]. Using a class of performance Petri nets (PPN), called deterministic and stochastic Petri nets (DSPN) [30], Bosch and Schmid [31] showed how the detailed understanding of the functionality of protocol mechanisms can be obtained.

3 Programming Language Models

3.1 Abstract Programs

The use of programming languages for specifying communication protocols is motivated by the observation that protocols are simply one kind of algorithm, and that programming languages provide a clear and concise way of describing algorithms. In an abstract program model, protocols are described as parallel programs. The example of using an abstract program to specify ABP is shown as follow:

```
                  /*******  Sender   *********/
        SEQ := 1                          /* initialization */
start:
        Data_SENT := INDATA[PT]           /* get next message to be sent */
        PT := PT + 1                      /* prepare PT for next message */
        SEQ := (SEQ + 1) MOD 2            /* switch seq # for the next message */
send:
        ACK := none                       /* erase previous ack */
        send(DATA_SENT)                   /* send message */
        wait(ACK)                         /* wait for acknowledgement */
        if (ACK == SEQ) then
                goto start                  /* next message */
        else goto send                      /* re-transmission */

                /*******  receiver *********/
        PSEQ# := 1                          /* initialization */
start:
        SEQ# := none                        /* erase previous sequence # */
        wait(DATA_RCV, SEQ#)                /* wait for a message */
        if (SEQ# == PSEQ#)
                then goto nack              /* send Nack */
        else begin
             OUTPUT(DATA_RCV)               /* output data received */
             PSEQ# := (PSEQ# + 1) MOD 1
            end
nack:
        send(PSEQ#)
        goto start                          /* next message */
```

Bochmann [32] made one of the earliest attempts at specifying and verifying a simple HDLC protocol using an abstract program. The protocol was specified in a free-style Pascal. The program structure is event driven and similar to a state-transition model in many ways. He partially verified the protocol by stating three safety invariants that described the number of messages sent and received by each protocol entity. However, the proof was not formal.

Stenning [33] also used an abstract program to specify and verify a data-transfer protocol. His code was very close to standard Pascal, which enabled him to rely on the standard Pascal rules for deriving pre- and post-conditions of the invariant assertions.

Krogdahl [34] developed the technique of *protocol skeletons* for specifying and verifying safety properties of classes of protocols. He attempted to provide as general a program specification as possible, using an Algol-like language. Ansart et al. [35] developed a protocol description and implementation language (PDIL) for specifying protocols and allowing automatic implementation. Based on standard Pascal, PDIL relieves the user of all the constraints of putting into a programming language form (e.g., the definition of data structures and procedures for manipulating typed objects). The latter work is done by a processor for PDIL, which generates coherent Pascal text. Castanet et al. [36] presented a methodology of using Ada for the specification and implementation of protocols. Compared with other programming languages, Ada has the advantage of homogeneity. However, its main drawback is in performance.

3.2 CSP

Hoare's Communicating Sequential Processes (CSP) [37] is a high-level concurrent language designed for distributed systems. A CSP program consists of a number of processes that do not share any common memory. Communications between processes are accomplished only through message passing. In CSP, guarded commands are used to describe nondeterministic behavior of each process. Major CSP constructs are described briefly as follows:

1. Parallel commands $[P_1 \| P_2 \| \cdots \| P_n]$ specify the concurrent execution of n processes $P_1, P_2, \cdots, P_n$.
2. Input command $P_j?(x)$ and output command $P_i!e$ specify the communication between processes P_i and P_j. (Process P_j sends the value of e to variable x of process P_i.)
3. Both alternative command

$$\begin{array}{l} [\\ b_1; I/0_1 \rightarrow \mathit{commandlist}_1 \mid \\ b_2; I/0_2 \rightarrow \mathit{commandlist}_2 \mid \\ b_n; I/0_n \rightarrow \mathit{commandlist}_n \mid \\ ] \end{array}$$

and repetitive command

$$
\begin{array}{l}
*[\\
b_1; I/0_1 \rightarrow commandlist_1 \mid\\
b_2; I/0_2 \rightarrow commandlist_2 \mid\\
b_n; I/0_n \rightarrow commandlist_n \mid\\
]
\end{array}
$$

are in the form of guarded commands.

A protocol in this model is represented by a CSP program, in which each protocol entity is represented by a process. As an example, a CSP specification of the ABP is shown in Figure 3.

3.3 CCS

Besides CSP, another abstract program model that has been receiving considerable attention in the literature is Milner's Calculus of Communicating Systems (CCS) [38]. CCS has a small but powerful set of operators for specifying the communication behavior of concurrent systems. In this model, a protocol is represented by a set of communicating agents. An agent is capable of communicating with other agents (via internal ports) or with an external observer of the system (via external ports). The basic notion in CCS is a set of atomic events denoting either internal events or communication events. These atomic events are represented as follows:

1. αx for input event, where x is a value variable.

2. $\overline{\alpha}e$ for output event, where $\overline{\alpha}$ is a label complementary to α, and e is a value expression.

3. τ for internal event.

Based on the occurrences of events, CCS has operators to express the following:

1. Sequences of events, by operator ".".

```
ABP::[Sender||Receiver]

Sender::
frame:record
        data:...;
        seq:(0,1)
      end;
DATA:...;
SEQ:(0,1);
Ack:(0,1);
done:boolean;
SEQ:=1;
*[User?(DATA) --> SEQ :=(SEQ+1) mod 2;
                  frame.data:=DATA;
                  frame.seq:=SEQ;
                  done:=false;
                  *[-done;Receiver!(frame)-->Receiver?(Ack);
                                           [Ack=SEQ --> done:=true; |
                                            Ack=(SEQ+1) mod 2 --> skip;
                                           ]
                   ]
]

Receiver::
frame:/*same as in Sender*/
exp:(0,1);
exp:=1;
*[Sender?(frame)-->[frame.seq=(exp+1)mod 2 --> User2!(frame.data);
                                               exp:=(exp+1)mod 2 |
                    frame.seq=exp-->skip
                   ];
                   Medium!(exp)
]
```

Figure 3: ABP Specified in CSP

Sender:

$$S \Leftarrow I.S_0$$
$$S_0 \Leftarrow \overline{\alpha}_0.\{\delta_0.I.S_1 + \delta_1.S_0\}$$
$$S_1 \Leftarrow \overline{\alpha}_1.\{\delta_1.I.S_0 + \delta_0.S_1\}$$

Receiver:

$$R_0 \Leftarrow \alpha_0.\overline{O}.\overline{\delta}_0.R_1 + \alpha_1.\overline{\delta}_1.R_0$$
$$R_1 \Leftarrow \alpha_1.\overline{O}.\overline{\delta}_1.R_0 + \alpha_0.\overline{\delta}_0.R_1$$

Figure 4: CCS Model of the ABP

2. Choice between sequences of events, by operator ”+”.

3. Recursion for specifying infinite sequences, by operator ”$\Leftarrow$”.

4. Parallel composition of agents to form systems of communicating agents, by operator ”$||$”.

5. Hiding of a subset of the internal ports, enabling one to abstract away from the internal details of an agent, by operator ”\”.

Formally, a protocol in CCS is defined by the following BNF notation:

$$t ::= x \mid op(t_1, t_2, \cdots, t_n) \mid x \Leftarrow t$$

where x is a variable name, op is an operator, and $(t_1, t_2, \cdots, t_n)$ is a CCS expression. As an example, a CCS specification of the ABP is shown in Figure 4. Note that in this example, α represents data flowing from Sender to Receiver, and δ represents acknowledgment flowing from Receiver to Sender, and I and O are communication actions with outside observers.

The global behavior of a protocol in the CCS model can be computed by applying the operation of parallel composition to all its communicating agents. For example, the ABP can be composed as follows: (S $||$ R_0). During parallel composition, pairs of events such as αx and $\overline{\alpha}e$ can be coupled and become a rendezvous event $x := e$.

One of the formal specification languages developed by ISO, called LOTOS, is heavily based on CCS. LOTOS will be discussed in Section 4.4.

3.4 Other Programming Language Models

Other programming language models to be described in the rest of this section include *temporal logic techniques* and *abstract data types.*

Temporal logic was first introduced by Pnueli [39] as an adaptation of a classical model logic suitable for defining the semantics of computer programs. Later, it was used by Hailpern and Owicki [40] and Schwartz and Melliar-Smith [41] to specify and verify the liveness (or progress) property of protocols. The liveness property requires that certain transitions eventually take place, and is difficult or impossible to state and prove in state-transition specifications, since conventional logic cannot refer to any state other than the present one.

Hailpern and Owicki [40] modeled a protocol system as a set of interacting modules that represent the logical units of the system. Both active (called process) and passive (called monitor) modules may be specified. They exploit this modularity in their specifications and proofs. They have verified the safety and liveness properties of the ABP, and Stenning's data-transfer protocol [42].

Schwartz and Melliar-Smith [41] developed specifications employing temporal logic with a more explicit notion of system state. They divided the task of specifying and verifying protocols into two parts: service level specification and network level specification. The service level defines the operations available to the users of the protocol, while the network level represents an abstract specification of the essential details of the protocol implementation. The goal is to verify the service level from the network level and to verify the network level from the protocol code. They illustrated the feasibility of their technique by formally verifying both safety and liveness properties of the ABP.

In 1984, Sabnani and Schwartz [43] verified a multidestination protocol, called the selective repeat procedure, for a satellite broadcast channel using a time-division multiplexed technique. This procedure is modeled as a parallel program in a Pascal-like language. The correctness of the parallel program model is shown using temporal logic. Both the safety and liveness properties are satisfied.

Another programming language model is abstract data types [44]. Abstract data types are an attempt to encapsulate data and the operations that manipulate it. There are two approaches in this area: abstract model and axiomatic. However, the distinction between these two approaches may not be so great in practice, since it is possible to write abstract model specifications in the axiomatic notation. It was reported by Sunshine [45, 46] that experience with these techniques is still limited. More research in this direction is required.

3.5 Comparison to State Transition Model

The major advantage of programming language models over state transition models is their capability to handle variables and parameters, such as sequence numbers and timers, which may take on values of a wide range. Another advantage is their ability to specify a whole protocol and most of its properties rather than only general correctness properties.

However, since protocol specification in programming language models may be very similar to protocol implementations, unessential features are often combined with the essential algorithms. In addition, efforts to prove the correctness of programs representing communication protocols far exceed those required to develop algorithms or programs. Program proof usually depends heavily on human ingenuity and intuition, and the automation of proof seems quite impossible and, therefore, is still far away from being of significant use.

4 Hybrid Models

4.1 EFSM

The extended finite state machine (EFSM) model is a generalization of the FSA model. The EFSM model allows multiple-state variables of various types; consequently, a state in this model becomes a vector of these variables and the transition function becomes more complex. The values of these state variables are changed by the occurrence of events. An event can occur only if certain enabling conditions are satisfied. An enabling condition is a predicate on the state variables. When more than one event in an event-driven system are enabled, any one of the enabled events is allowed to occur, resulting in nondeterminism.

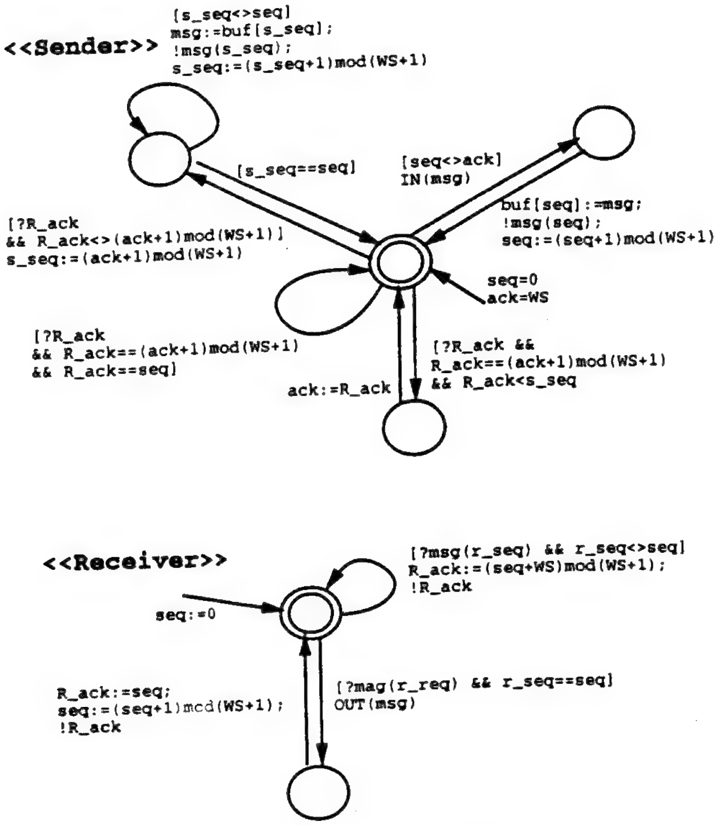


Figure 5: ABP in the EFSM Model

As in the FSA model, in the EFSM model, each protocol is represented by a set of communicating protocol entities. The communication is done by message passing through communication channel between protocol entities. Formally, a protocol entity is defined as follows [47]:

An *EFSM Protocol Entity* P_x is an eight-tuple $(q, W, Val, \Sigma, \lambda, \delta, i, f)$, where:

1. q is the set of states in P_x.

2. W is the finite set of variables used in P_x. The values of the variables in W may change only as the result of the execution of some transitions and/or time change.

3. Val is an n-tuple, where n is the number of variables in W and Val represents the initial values of the variables in W.

4. Σ is the finite set of service transitions in P_x.

5. λ is the finite set of peer transitions in P_x. Note that a (peer or service) transition in P_x consists of two parts: Predicate and Action. A predicate is a combination of some mathematic operations on the variables in W and represents the required conditions that the entity has to be in for the corresponding action part of the transition to be executable.

6. δ is the transition function of P_x which maps $q \times (\Sigma \cup \lambda)$ into q; if $\delta(s_1, t) = s_2$, where $s_1 \in q$, $s_2 \in q$, and $t \in (\Sigma \cup \lambda)$, s_1 is called the starting state of t and s_2 is the ending state of t. Note that a transition t_r is said to be executable only when the current state of P_x is the starting state of t_r and the predicate condition is fully met. In addition to the aforementioned conditions, for a *receive/read* transition t_r to be executable, the *message to be received/read* must be already in the incoming channel to P_x.

7. i is the initial state of P_x.

8. f is the set of the final states in P_x. Note that for a typical non-terminating protocol, f usually has only one element, i, which is the initial state of the entity.

Note that in this definition, a distinction is made between two kinds of transitions executed in the entities, namely, *Peer Transitions* that support the interaction between two peer protocol entities and *Service Transitions* that perform the interaction between a protocol entity and its environment (e.g. the protocol service users) through the SAP (service access point).

An example that shows how to apply EFSM to specify the ABP is given in Fig: 5. In this example, a predicate is enclosed in a pair of square brackets. In both predicate and action parts of a transition, ”!” and ”?” are used to denote sending of a message or acknowledgement and reception of a message or acknowledgement, respectively.

Several researchers have proposed particular forms of such EFSM models for specifying protocols. While differing in names and in the details of syntax, they may be considered equivalent in expressive power.

Based on Keller's transition model for parallel programs [48], Bochmann has proposed a general transition model for the formal specification of protocols, the specification of the services provided, and the verification of the correct operations [49]. He discussed these issues by considering, as an example, the HDLC classes of procedures.

In 1989, Chu [50] extended the TG model, called the extended transmission grammar (ETG) model, to contain context variables such as sequence numbers in protocol specifications. The notion of communicating sequential processes (CSP) [37] in using ”?” and ”!” as the receive and send events, respectively, is used. In this context, ”?” is a blocking receive as it is in CSP; whereas ”!” is a non-blocking send that will not wait for the communicating partner to be ready for the send event to be executable, as is required for ”!” in CSP.

Shankar and Lam [51] used an event-driven process model to specify a version of the HDLC protocol between two communicating protocol entities. The protocol is verified using the method of *projections* [52]. The verification serves as a rigorous exercise to demonstrate the applicability of this method to the analysis of realistic protocols. This procedure has been automated.

4.2 SDL

The Specification and Description Language (SDL) [53, 54] was developed by study groups SGXI and SGX of the CCITT. It is meant specifically for the specification and design of the telecommunications systems, such as telephone switches. The study was started in 1968. Since

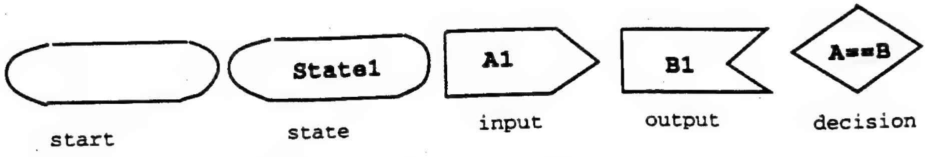


Figure 6: Basic Constructs of SDL

then, it has evolved from an informal drawing technique to a formal description technique. There are two variants of SDL in use: a graphical form which is more popular, and a textual form.

SDL provides the concept of *system* as a boundary between the specification and the environment. The behavior of a system is constituted by the combined behavior of a number of processes in the system. The cooperation between the processes is performed asynchronously by discrete messages, called signal. The concept of block is provided as a way to structure a system. One or more processes are grouped into a block. From a modeling point of view, a system is made up of blocks that interconnect with each other and with the environment by channels. A channel is a means of conveying signals.

In SDL, a process is an EFSM. There are five basic constructs for the description of a process: start, state, input, output and decision as shown in Figure 6. SDL also allows abstract data type to be used in the specification. Figure 7 describes a version of ABP in SDL. In this specification, the sending of signal of type *msg* in the sender matches with the corresponding receiption of signal of type *msg*. This, thus, accomplishes the following two assignments: $i := o$ and $a := seq$. It happens to signal of type *ack*, too.

The most recent extensions to the language are in the area of object-orientation which are included in SDL-92 [55]. In SDL-92, a clear distinction between types and instances, specialization of types into subtypes, and the concept of generic types are introduced. This new language is backward compatible.

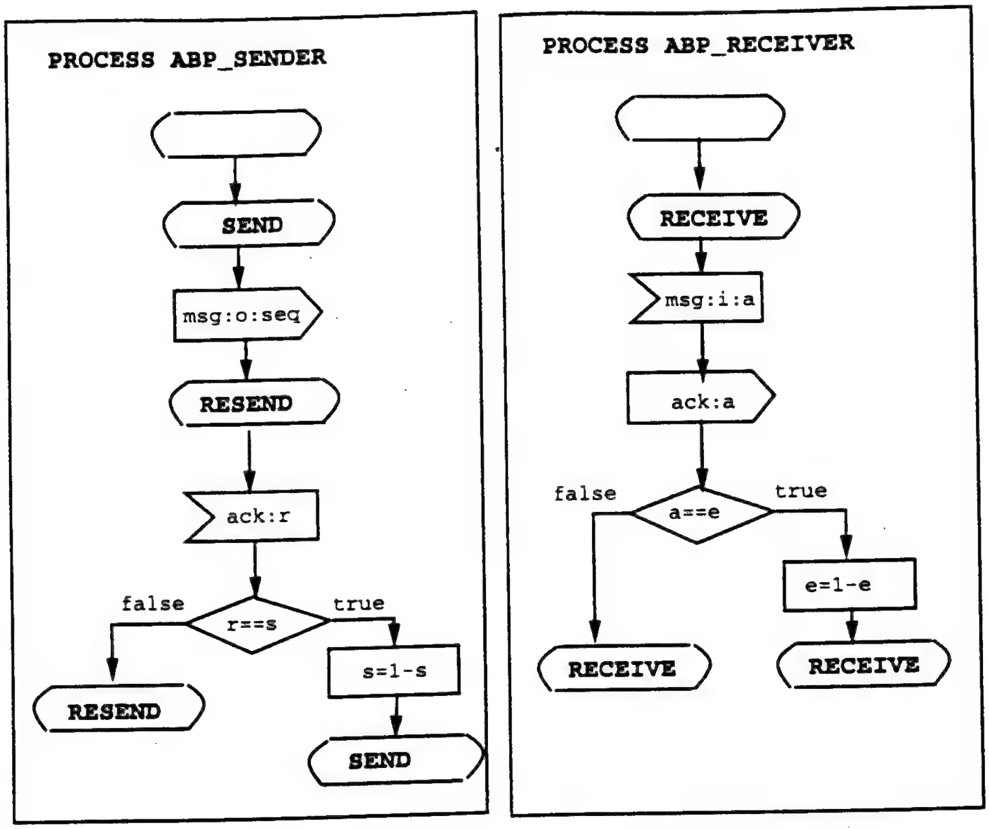


Figure 7: ABP written in SDL

4.3 Estelle

Estelle [56] is an FDT standardized by ISO. It can be used to describe distributed concurrent information processing systems. The language is based on the concept of EFSM. A distributed system specified in Estelle is viewed as a collection of communicating components called tasks. Tasks are instances of modules. Modules may be nested and organized in a hierarchical structure. Each module has a number of input/output access points called interaction points. A task can send a message (is called an interaction) to another task through a previously established communication channel between two interaction points of the tasks. The internal behavior of a module is described in terms of a nondeterministic communicating state automaton whose transition actions are given in the form of Pascal statements (with some restrictions and extensions). Each transition begins with the keyword *trans*. A transition declaration composed of two parts: the transition condition and the transition action. The transition condition is composed of one or more clauses of the following categories:

from-clause: (*from* $A_1, A_2, \cdots, A_n$, where A_i is a state identifier)

when-clause (*when p.m*, where p is an interaction point and m is an interaction)

provided-clause (*provided* B, where B is a boolean expression)

priority-clause (*priority* n, where n is a non-negative constant)

delay-clause (*delay*(E_1,E_2), where E_1 and E_2 are non-negative integer expressions)

Transitions without a when-clause are called spontaneous transitions. A spontaneous transition with a delay-clause is called a delay transition.

Besides containing a transition block of a sequence of Pascal statements between *begin* and *end*, a transition action may also have a to-clause to specify the next state which will be attained once this transition is fired. Without a to-clause, a transition's next state is its current state.

The following is a specification of ABP in Estelle. Note that all modules are defined in a principal module called *specification* module.

```
specification ABP systemprocess;
    { This is the top level module body (specification).
      The specification has the attribute system process
      and all its children ( sender, receiver) are processes. }


    default individual queue;


    type
        MsgType=...;


    channel NChannel(Sender,Receiver);
        by Sender:
    msg(M:MsgType; Seq:integer);
by Receiver:
    ack(Seq:integer);


    channel SServ(User,Provider);
        by User:
    send(M:MsgType);


    channel RServ(User,Provider);
        by Provider:
    del(M:MsgType);


    { Module header definitions }


    module SenderType process;
```

```
     ip u:SServ(Provider) common queue;
  s:NChannel(Sender) individual queue;
  end; { of SenderType }

  module ReceiverType process;
     ip u:RServ(Provider) common queue;
  s:NChannel(Receiver) individual queue;
  end; { of ReceiverType }

  { The body of the Sender }

  body SenderBody for SenderType;

type
BufferType=...;

     var
b : integer;
Msg : MsgType;
UnAckedMsgs: BufferType;

     state SS1,SS2;

procedure Store(var B:BufferType; var M:MsgType);
primitive;

procedure Retrieve(var B:BufferType; i:integer; Var M:MsgType);
primitive;

{ Initialization part for SenderBody }
```

```
initialize

to SS1 { S is the initial state }
    begin b := 0; end;

{ Transition declaration part for SenderBody }

trans

from SS1
    to SS2
        when u.send
        begin
output s.msg(M,b);
Store(UnAckedMsgs,M);
end;

from SS2
    to SS2
when n.ack
   provided Seq <> b
        begin
Retrieve(UNAckedMsgs,Msg);
output s.msg(Msg,b);
end;

    to SS1
when n.ack
    provided Seq = b
```

```
     begin
b := 1 - b;
end;

    end;  { of SenderBody }

    body ReceiverBody for ReceiverType;
     { Body for ReceiverType }
        var
    b : integer;

state SR;

initialize

to SR
    begin b := 0; end;

trans

from SR
    to SR
when n.msg
    provided b=Seq
     begin
     output u.del(M);
output s.ack(Seq);
b := 1 - b;
     end;
     to SR
 when n.msg
     provided b<>Seq
         begin output s.ack(Seq); end;
     end; { of ReceiverBody }

     { Declare IPs for the specification (the outermost module). }

     modvar
         { Modvars are the realization of modules.  This declaration
   declares how each modvar "looks like," i.e., to define its
   type.  The content of each module is associated with the
   module bodies by the "init" statements below }

         SendEntity:SenderType;
RecEntity:ReceiverType;

    initialize
        begin
{ Associates modvars to their body }
init SendEntity with SenderBody;
init RecEntity with ReceiverBody;

{ Connecting and attaching channels }
connect SendEntity.s to RecEntity.s;

end;

end. {Specification}
```

There are a lot of works in developing toolsets for designing protocols using Estelle. For instance, Estelle Development Toolset (EDT) [57] provides tools for compiling and simulating Estelle protocols specifications, and Estelle Simulator based on an Interpretative Machine (ESTIM) [58] is a tool for validating protocols. Recently, some visualization tools are also being developed for the Estelle. A tool called GROPE [59] can provide a dynamic, graphical representation of a simulated Estelle specification. GROPE animates the firing of Estelle transitions and exchange of interactions over channels.

4.4 LOTOS

The Language of Temporal Ordering Specification (LOTOS) [60, 61] was adopted as ISO international standard IS8807 in 1989. The basic idea from which LOTOS was developed is that systems can be specified by defining temporal relation among the interactions that constitute the externally observable behavior of a system. Contrary to what its name seems to suggest, LOTOS is not related to temporal logic, but it is based on process algebraic methods. CSP and CCS are also methods of process algebra.

A LOTOS specification describes a system via a hierarchy of process definitions. A process is an entity able to perform internal actions and to interact with other processes via interaction points called gates. Complex interactions between processes are built up from elementary units of synchronization which is called an event. The constructs of LOTOS are similar to those of CSP and CCS. LOTOS has the following major constructs:

Inaction: *stop*

A completely inactive process.

Action Prefix: *g;B*

This is a unary prefix operator which produces a new behavior expression out of an existing one, by prefixing the latter with an action followed by a semicolon.

Choice: B_1 *[]* B_2

It denotes a process that behaves either like B_1 or B_2.

Parallelism: B_1 | $[g_1, g_2, \cdots, g_n]$ | B_2

The set S=$\{g_1, g_2, \cdots, g_n\}$ is a set of synchronization gates. This implies that when process B_1 is ready to execute some actions at one of the synchronization gates, it is forced, in the absence of alternative actions, to wait until B_2 offers the same action. In the case that the set of synchronization gates is empty, the construct is pure interleaving. On the other hand, when the set is all the gate, it is full synchronization.

Hiding: Hiding allows one to transform some oberservable actions of a process into unobservable ones.

A version of LOTOS specification for ABP is shown in Figure 8.

The major advantage of LOTOS in comparing to other two standard FDT, SDL and Estelle, is that it can allow designer to reason formally about behavior since the algebras that it is based define a rigorous set of transformation rules and equivalence relations. However, this benefit also has its short fall since its mathematical nature makes it harder to use than the other two. In the past few years, collaborated affords from ISO and CCITT have been made to extend LOTOS with graphical syntax. G-LOTOS which is currently being standardized is intended to provide a better readability and more intuitive understanding of formal specifications than textual LOTOS. G-LOTOS has the same semantic as the textual LOTOS; an introduction to G-LOTOS can be found in [62].

4.5 Others Hybrid Models

There are other hybrid models that were proposed for protocol specification. The language IBM uses for describing SNA is called format and protocol language (FAPL). It is derived from PL/1 and contains additional constructs for handling FSMs and processes. A protocol specified in this form is precise, readily accessible to the implementing product designers and programmers, and structurally close to the implementations [63, 64, 65].

```
specification M_ABP[in,out] :=
   hide S_Chan in
        Sender[in,S_Chan](0) | [S_Chan] |
        Receiver[SL_Chan,out](0)

where
        process Sender[in,S_Chan](seq_cnt):=
                in ? Msg:MsgType  --> Transmit[S_Chan](Msg,seq_cnt)
        where
                process Transmit[S_Chan](Msg,seq_cnt):=
                        S_Chan ! Msg ! seq_cnt;
                        S_Chan ? Seq:integer -->
                               (  (Seq == seq_cnt) -->
                                       Sender[in,S_Chan](1 - seq_cnt)
                               [] (Seq != seq_cnt) -->
                                       S_Chan ! Msg ! seq_cnt;
                                       Transmit[S_Chan](Msg,seq_cnt)
                               )
                endproc
        endproc

        process Receiver[S_Chan,out](rec_cnt)
                S_Chan ? Msg:MsgType ? Seq:integer -->
                        (  (Seq == rec_cnt) -->
                                out! Msg;
                                S_Chan ! rec_cnt;
                                Receiver[S_Chan,out](1-rec_cnt);
                        [] (Seq != rec_cnt) -->
                                R_Chan ! (1-rec_cnt);
                                Receiver[R_Chan,out](rec_cnt,Wsize)
                        )
        endproc
endspec
```

Figure 8: ABP Specified in LOTOS

Aggarwal et al.[66] has proposed the selection/resolution model for specifying, analyzing and validating the behavior of protocols. The model centers around abstract entities called processes. Parallelism is addressed directly with concurrent transitions in each process dependent on the status of neighboring process. Protocol specification is accomplished by defining many small and easy to specify interacting processes, which collectively describe the behavior of the protocol. The feasibility of the model was demonstrated by applying it to the ABP and a file-transfer protocol [67].

Krumm and Drobnik [68, 69] proposed the CIL (Communication Service Implementation Language) approach for the development of communication services. It is based on an event-oriented model of program execution and a first-order predicate calculus. The verification of a program written in CIL is supported by the automated generation of program axioms and by the predicate calculus.

Meer et al. [70] introduced algebraic specification methods based on language ACT ONE [71]. The algebraic methods model abstract data types as many sorted algebras. These sorted algebras are used as mathematical objects and serve as a semantic domain of the system being modeled. One advantage of this approach is that the mathematical objects provide a precise semantics to its specification.

5 Conclusion

Using FDTs in protocol specification facilitates the development of unambiguous, complete and correct protocol specifications. It also forms the basis for automated protocol verification, semi-automated implementation of protocols and automatic generation of test sequences for the conformance testing of a protocol implementation against its standard.

Various FDTs and their models have been described in this paper. Among them, FSA is one of the earliest models and has been used extensively in areas such as formal protocol verification, conformance testing, etc. However, due to FSA's severe restrictions as discussed in Section 2.1, researchers have been searching for better models. For example, efforts within ISO and CCITT have been made since the late 70s to develop standardized FDTs. For these efforts, hybrid models have received the most attention. In fact, after approximately more than a decade of hard work, three complementary standardized FDTs prevail in their full status of international standards (IS) or recommendation: Estelle (IS9074) and LOTOS (IS8807) defined by ISO, and SDL (Z100) by CCITT. These FDTs have the full expressive power and abstraction capabilities to allow for the complete architectural specification of OSI protocols and services as well as the specification of OSI architecture concepts and constructions at appropriate levels of abstraction.

Since the standardization of these FDTs, enhancements and many supporting tools that facilitate the use of them in protocol specification have been developed. However, the research on how these standard FDTs can be applied to other areas of protocol engineering such as protocol conversion and synthesis is just beginning. To gain worldwide acceptance, future research on FDTs should focus on broadening the applications of these standard FDTs to other areas of protocol engineering.

References

[1] M. T. Liu, "Protocol Engineering," in *In Advances in Computers* (M.C. Yovits, ed.), vol. 27, pp. 79–195, New York: Academic Press, July 1989.

[2] S. Vuong, "Guest Editorial of Special Issue on FDT," *Computer Networks and ISDN Systems*, vol. 23, pp. 323–324, 1992.

[3] S. Consortium and J. Bruijning, "Evaluation and Integration of Specification Languages," *Computer Networks and ISDN Systems*, vol. 13, pp. 75–89, 1987.

[4] D. Sidhu, A. Chung, and T. Blumer, "Experience with Formal Methods in Protocol Development," *Computer Communication Review*, vol. 21, pp. 81–101, Apr. 1991.

[5] W. C. Lynch, "Reliable Full-duplex Transmission over Half-duplex Telephone Lines," *Comm. of ACM*, vol. 11, no. 6, pp. 362–372, 1968.

[6] K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson, "A Note on Reliable Full-duplex Transmission over Half-duplex Lines," *Comm. of ACM*, vol. 12, no. 5, pp. 260–261, 1969.

[7] G. V. Bochmann, "Finite State Description of Communication Protocols," *Computer Networks*, vol. 2, pp. 362–372, 1978.

[8] C. H. West and P. Zafiropulo, "Automated Validation of a Communication Protocol: The CCITT X.21 Recommendations," *IBM J. Research and Development*, vol. 22, no. 1, pp. 60–71, 1978.

[9] M. G. Gouda and Y. T. Yu, "Synthesis of Communicating Finite-State Machines with Guaranteed Progress," *IEEE Transactions on Communications*, vol. COM-32, no. 7, pp. 779–788, 1984.

[10] M. G. Gouda and K. S. The, "Modeling Physical Protocols Using Communicating Finite State Machines," in *Proc. ACM/IEEE Data Comm. Symp. 9th*, pp. 54–62, 1985.

[11] M. G. Gouda and C. K. Chang, "A Technique for Proving Liveness of Communicating Finite State Machines with Examples," in *Proc. ACM Symp. on Principles of Distributed Computing. 3rd*, pp. 38–49, 1984.

[12] T. T. Lee and M. Y. Lai, "A Relational Algebraic Approach to Protocol Validation," *IEEE Transactions on Software Engineering*, vol. SE-14, no. 2, pp. 184–193, 1988.

[13] J. Harangozo, "An Approach to Describing a Link Level Protocol with a Formal Language," in *Proc. ACM/IEEE Data Comm. Symp. 5th*, pp. 4.37–4.49, 1977.

[14] A. Y. Teng and M. T. Liu, "A Formal Approach to the Design and Implementation of Network Communication Protocols," in *Proc. IEEE COMPSAC*, pp. 722–727, 1978.

[15] A. Y. Teng and M. T. Liu, "A Formal Model for Automatic Implementation and Logical Validation of Network Communication Protocols," in *Proc. IEEE Computer Networking Symp.*, pp. 114–123, 1978.

[16] A. Y. Teng and M. T. Liu, "The Transition Grammar Model for Protocol Construction," in *Proc. NBS Trends and Applications Conf.*, pp. 110–120, 1980.

[17] C. S. Lu, *Automated Validation of Communication Protocols. Ph.D. Dissertation.* Columbus, Ohio: The Ohio State University, 1986.

[18] L. D. Umbaugh and M. T. Liu, "A Comparison of Communication Protocol Validation Techniques," in *Proc. IEEE Int. Conf. Comm.*, pp. 4C.4.1–4C.4.7, 1982.

[19] L. Umbaugh, *Automated Techniques for Specification and Validation of Communication Protocols. Ph.D. Dissertation.* Columbus, Ohio: The Ohio State University, 1983.

[20] L. D. Umbaugh, M. T. Liu, and C. J. Graff, "Specification and Validation of the Transmission Control Protocol Using Transmission Grammar," in *Proc. IEEE COMPSAC*, pp. 207–216, 1983.

[21] M. Diaz, "Modeling and Analysis of Communication and Cooperation Protocols Using Petri net based Models," *Computer Networks*, vol. 6, no. 6, pp. 419–441, 1982.

[22] P. M. Merlin, "A Methodology for the Design and Implementation of Communication Protocols," *IEEE Transactions on Communications*, vol. COM-24, no. 6, pp. 614–621, 1976.

[23] J. Postel and D. J. Farber, "Graph Modeling of Computer Communication Protocols," in *Proc. Texas Conf. on Computing Systems. 5th*, pp. 66–77, 1976.

[24] F. J. W. Symons, "The Verification of Communication Protocols Using Numerical Petri nets," *Australian Telecommunication Research*, vol. 14, no. 1, pp. 34–38, 1980.

[25] W. Jurgensen and S. T. Vuong, "Formal Specification and Verification of ISO Transport Protocol Components Using Petri nets," in *Proc. ACM SIGCOMM 84*, pp. 75–82, 1984.

[26] R. R. Razouk and G. Estrin, "Modeling and Verification of Communication Protocols in SARA: the X.21 Interface," *IEEE Transactions on Computers*, vol. C-29, no. 12, pp. 1038–1052, 1980.

[27] R. R. Razouk, "Modeling X.25 Using the Graph Model of Behavior," in *Proc. IFIP Int. Workshop on PSTV. 2nd*, pp. 197–214, 1982.

[28] B. Berthomieu and M. Menasche, "An Enumerative Approach for Analyzing Timed Petri nets," in *Proc. IFIP Information Processing*, pp. 41–46, 1983.

[29] J. Sifakis, "Use of Petri Nets for Performance Evaluation," in *Proc. 3rd Int. Workshop on Measuring Modeling and Evaluating Computer Systems*, pp. 75–93, 1977.

[30] M. A. Marsan and V. Signore, "Timed Petri Net Performance Models of Fiber Optics LAN Architecture," in *Proc. Int. Workshop on Petri Nets and Performance Models*, pp. 66–74, 1987.

[31] M. Bosch and G. Schmid, "Generic Petri net Models of Protocol Mechanisms in Communication Systems," *Computer Communications*, vol. 14, pp. 143–156, Apr. 1991.

[32] G. V. Bochmann, "Logical Verification and Implementation of Protocols," in *Proc. ACM/IEEE Data Comm. Symp. 4th*, pp. 8.5–8.20, 1975.

[33] V. N. Stenning, "A Data Transfer Protocol," *Computer Networks*, vol. 1, no. 2, pp. 99–110, 1976.

[34] S. Krogdahl, "Verification of a Class of Link-Level Protocols," *BIT*, vol. 18, pp. 436–448, 1978.

[35] J. P. Ansart, "GENEPI/A - A Protocol Independent System for Testing Protocol Implementation," in *Proc. IFIP Int. Workshop on PSTV. 2nd*, pp. 523–528, 1982.

[36] R. Castanet, A. Dupeux, and P. Guitton, "ADA, a Well Suited Language for Specification and Implementation of Protocols," in *Proc. IFIP Int. Workshop on PSTV. 5th*, pp. 247–258, 1985.

[37] C. A. R. Hoare, "Communicating Sequential Processes," *Comm. of ACM*, vol. 21, no. 8, pp. 666–677, 1978.

[38] R. Milner, *A Calculus of Communicating System.* Berlin, W. Germany: Springer-Verlag, 1980.

[39] A. Pnueli, "The Temporal Logic of Programs," in *Proc. IEEE/ACP Symp. on Foundations of Computer Science, 18th*, pp. 46–57, 1977.

[40] B. T. Hailpern and S. Owicki, "Verifying Network Protocols Using Temporal Logic," in *Proc. NBS Trends and Applications Conf.*, pp. 18–28, 1980.

[41] R. L. Schwartz and P. M. Melliar-Smith, "Temporal Logic Specification of Distributed Systems," in *Proc. IEEE Int. Conf. on DCS. 2nd*, pp. 446–454, 1981.

[42] B. T. Hailpern and S. Owicki, "Verifying Network Protocols Using Temporal Logic," *IEEE Trans. Comm.*, vol. COM-31, no. 1, pp. 56–68, 1983.

[43] K. Sabnani and M. Schwartz, "Verification of a Multidestination Selective Repeat Procedure," *Computer Networks*, vol. 8, pp. 463–478, 1984.

[44] J. V. Guttag, *The Specification and Application to Programming of Abstract Data Types.Ph.D. Dissertation.* Toronto, Canada: University of Toronto, 1975.

[45] C. A. Sunshine, "Experience with Four Automated Verification Systems," in *Proc. IFIP Int. Workshop on PSTV. 2nd*, pp. 373–380, 1982.

[46] C. A. Sunshine, "Experience with Automated Protocol Verification," in *Proc. IFIP Int. Workshop on PSTV. 3rd*, pp. 229–236, 1983.

[47] H. J. Jeng and M. T. Liu, "From Service Specification to Protocol Converter: A Synchronizing Transition Set Approach," *Accepted for publishing in Computer Networks and ISDN Systems*, 1992.

[48] R. M. Keller, "Formal Verification of Parallel Programs," *Comm. of ACM*, vol. 19, no. 7, pp. 371–384, 1976.

[49] G. V. Bochmann, "A General Transition Model for Protocols and Communication Services," *IEEE Tran. Comm.*, vol. COM-28, no. 4, pp. 643–650, 1980.

[50] L. Umbaugh, *Towards Automating Protocol Synthesis and Analysis. Ph.D. Dissertation.* Columbus, Ohio: The Ohio State University, 1989.

[51] A. U. Shankar and S. S. Lam, "An HDLC Protocol Specification and Its Verification Using Image Protocols," *ACM Trans. Computer Systems*, vol. 1, no. 4, pp. 331–368, 1983.

[52] S. S. Lam and A. U. Shankar, "Protocol Verification via Projections," *IEEE Trans. Software Engineering*, vol. SE-10, no. 4, pp. 325–342, 1984.

[53] F. Belina and D. Hogrefe, "CCITT SDL: Overview of the Language and its Applications," *Computer Networks and ISDN Systems*, vol. 13, pp. 65–74, 1987.

[54] F. Belina and D. Hogrefe, "The CCITT-Specification and Description Language SDL," *Computer Networks and ISDN Systems*, vol. 16, pp. 311–341, 1989.

[55] O. Fargemand and A. Olsen, "Introduction to SDL-92," *Computer Networks and ISDN Systems*, vol. 26, pp. 1143–1167, 1994.

[56] S. Budkowski and P. Dembinski, "An Introduction to Estelle: A Specification Language for Districuted Systems," *Computer Networks and ISDN Systems*, vol. 14, pp. 3–23, 1987.

[57] S. Budkowski, "Estelle Development Toolset (EDT)," *Computer Networks and ISDN Systems*, vol. 25, pp. 63–82, 1992.

[58] J.-P. Courtiat and P. de Saqui-Sannes, "ESTIM: An Integrated Environment for the simulation and verification of OSI protocols specified in Estelle," *Computer Networks and ISDN Systems*, vol. 25, pp. 63–82, 1992.

[59] P. D. Amer and D. New, "Protocol Visualization in Estelle," *Computer Networks and ISDN Systems*, vol. 25, pp. 741–760, 1993.

[60] T. Bolognesi and E. Brinksma, "Introduction to the ISO Specification Language LOTOS," *Computer Networks and ISDN Systems*, vol. 14, pp. 3–23, 1987.

[61] P. van Eijk and et al., *The Formal Description Technique: LOTOS.* North-Holland, 1989.

[62] T. Bolognesi, E. Najm, and P. A. J. Tilanus, "G-LOTOS: A Graphical Language for Concurrent Systems," *Computer Networks and ISDN Systems*, vol. 26, pp. 1101–1127, 1994.

[63] S. C. Nash, "Automated Implementation of SNA Communication Protocols," in *Proc. IEEE Int. Conf. Comm.*, pp. 1316–1322, 1983.

[64] D. P. Pozefsky and F. D. Smith, "A Meta-implementation for System Network Architecture," *IEEE Tran. Comm.*, vol. COM-30, no. 6, pp. 1348–1355, 1982.

[65] S. C. Nash, "Formal and Protocol Language (FAPL)," *Computer Networks and ISDN Systems*, vol. 14, no. 1, pp. 61–77, 1987.

[66] S. Aggarwal, R. P. Kurshan, and K. Sabnani, "A Calculus for Protocol Specification and Validation," in *Proc. IFIP Int. Workshop on PSTV. 3rd*, pp. 19–34, 1983.

[67] S. Aggarwal and K. Sabnani, "Formal Specification of a File Transfer Protocol," in *Proc. IEEE INFOCOM*, pp. 47–57, 1986.

[68] H. Krumm and O. Drobnik, "Specification, Implementation and Verification of Communication Services on the Basis of CIL," in *Proc. IFIP Int. Workshop on PSTV. 3rd*, pp. 301–316, 1983.

[69] H. Krumm and O. Drobnik, "Transformation of Constructive Specifications of Services and Protocols into the Logical Language of CIL," in *Proc. IFIP Int. Workshop on PSTV. 4th*, pp. 31–46, 1984.

[70] J. d. Meer, R. Roth, and S. Vuong, "Introduction to Algebraic Specifications Based on The Language ACT ONE," *Computer Networks and ISDN Systems*, vol. 23, no. 5, pp. 363–392, 1992.

[71] H. Ehrig and B. Mahr, *Fundamentals of Algebraic Specification 1, Equations and Initial Semantics*, vol. 6. Springer, Berlin: EATCS Monographs on Theoretical Computer Science, 1985.

E. **L. S. Koh and M. T. Liu,**
"Test Path Selection Based on Effective Domains,"
Proc. 1994 IEEE Int'l Conf. on Network Protocols,
pp. 64–71, October 1994.

Test Path Selection Based on Effective Domains*

Liang-Seng Koh and Ming T. Liu

Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210-1277, USA

Abstract

In this paper, a method is proposed to produce test paths that check both data flow and control flow for a protocol specified in the Extended Finite State Machine (EFSM) model. The method first identifies a set of paths from a given specification to cover a dataflow selection criterion, then it appends state check sequences to some transitions in this set of paths for checking control flow. The criterion that our method employs for selecting these transitions is called effective domain for testing. Effective domain for testing is used to evaluate how effective a transition can be tested in a given path in terms of the range of values that the variables in this transition can have. Since each transition can appear in several paths, our method is to append state check sequences to its occurrences that have distinct effective domains. In addition, our method will compute the path domain for each path and make some inexecutable paths executable.

1 Introduction

Conformance testing aims at demonstrating that a protocol Implementation Under Test (IUT) conforms to the specification that it implements. Since an IUT is treated as a blackbox, test sequences that are sequences of input/output are needed for a tester to infer the properties of the IUT. For the purpose of generating test sequences, paths that are believed to be most error revealing must be selected from the corresponding specification. In the case that the specification is written in the Finite State Machine (FSM) model, the paths are, indeed, the test sequences. However, if the specification is specified in the Extended Finite State Machine (EFSM) model, test sequences will need to be constructed from these paths by substituting appropriate values for the input variables of each path.

In the literature, protocols specified in the FSM are always tested by making sure that there is no control flow errors associated with each transition. For such a purpose, state check sequences such as Unique Input/Output (UIO) [1] and Characterizing set (W-set) [2] are employed to identify the tail state of a transition. However, these techniques are inadequate for generating paths that can effectively test a protocol specified in the EFSM since the model describes not only the control aspect of the protocol, but also the data aspect of the protocol. For testing an IUT specified in the EFSM, some methods [3, 4, 5] have also been proposed. These methods employ the techniques of dataflow analysis to analyze the data interactions among the variables and then to choose paths from the specification to cover these interactions. Although the data flow of a protocol can be effectively examined by these methods, the control flow is ignored by these methods. In addition, these methods only concern with the issue of path selection; two other important issues, namely deciding path executability and identifying path domains for each selected path, are not addressed.

In this paper, a method is proposed to combine data flow testing with control flow testing. In our method, a set of paths is first selected to fulfill certain dataflow selection criterion, then state check sequences are appended to the transitions in the set for checking control flow errors. As a transition can appear in several paths, *effective domain for testing* is introduced as a criterion for evaluating the occurrences of a transition. The concept of effective domains is employed in this paper as a criterion for measuring how extensive a transition can be tested in a given path in terms of the range of possible values that the variables of this transition can have in the path. For each transition, our method will append state check sequences to its occurrences that have distinct effective domains. In addition to generating test paths for testing both control flow and data flow, our method also generates path domains for the respective test paths and makes some inexecutable test paths executable. As mentioned above, these two problems are very important problems; however, hardly any in depth discussions are provided in

*Research reported herein was supported by U.S. Army Research Office, under contracts No. DAAL03-91-G-0093 and No. DAAL03-92-G-0184. The views, opinions, and/or findings contained in this paper are those of the authors and should not be construed as an official Department of the Army position, policy or decision.

0-8186-6685-4/94 $4.00 © 1994 IEEE

the literature.

This paper is organised as follows: In Section 2, a model of the EFSM and the concept of dataflow testing are introduced. For ease of presenting our approach in the subsequent sections, a dataflow criterion called *all-du-paths* criterion is described in detail. The concept of effective domain for testing is discussed in Section 3. Then, assuming that a set of paths for covering all-du-paths criterion is given, a procedure for path selection is developed in Section 4. In Section 5, two related works recently published in the literature are compared with our work.

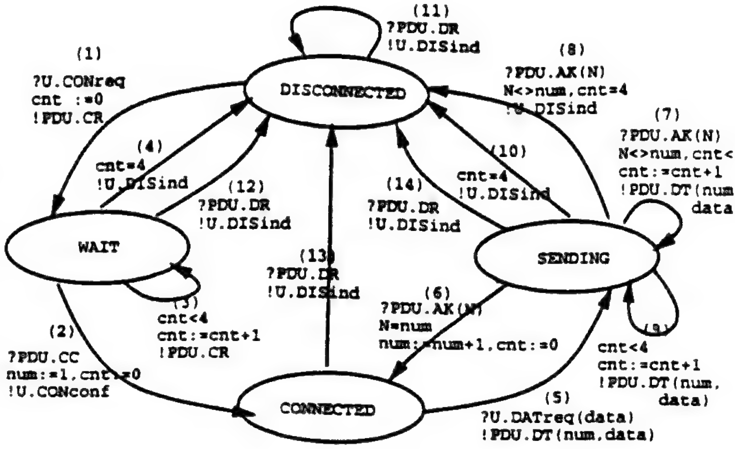


Figure 1: The INRES protocol

2 Background

2.1 Extended Finite State Machine

The EFSM model extends the FSM model by using variables to describe the data aspect of a protocol. In this paper, a protocol entity is modeled as an EFSM that is specified by a state-transition graph. In the graph, a state is represented by a vertex and a transition by a directed edge. Furthermore, it is assumed that every transition in the EFSM includes the following four parts that will be executed in this order: input, enabling conditions, program segment, and output.

Each of the four parts can have zero or more statements. Note that the order of statements given in each part must be observed. The input and output parts specify the interaction of a protocol with its environment. An input statement is prefixed by a '?' and an output statement is prefixed by a '!'. The enabling conditions part specifies the conditions under which a transition is fired when all the conditions are evaluated to be true. Thus, an enabling condition is a conjunctive predicate of all the conditions and is written as $p_1, p_2, \cdots, p_n$ where each p_n is a condition. Note that if the enabling conditions part has no statement, a transition can always be executed. The program segment is a sequence of assignment statements specifying the actions to be performed by a transition. An example of the INRES protocol [6] specified using this model is shown in Figure 1.

2.2 Data Flow Testing

In software engineering, dataflow testing is concerned with testing the data interactions in a program. Strategies based on dataflow testing, which are normally referred as dataflow selection criteria, look at interactions involving definitions for program variables and subsequent references that are affected by these definitions and require that certain such interactions be tested. Several of these criteria have been used in conformance testing for testing the data aspect of a protocol. For instance, the methods proposed in [3, 4] use *all-du-paths* [7]; whereas, the method proposed in [5] uses *all-simple-OI-paths* [8]. Since our method is to augment a set of paths generated to cover a dataflow selection criterion with some subsequences to cover the control aspect of a protocol, any of these dataflow criteria can be used in our method for selecting a set of paths to cover the data interactions. For ease of discussing our algorithm, the all-du-paths is used in this paper.

Before discussing the all-du-paths criterion, several terms need to be defined. In dataflow terminology, a variable, x, is said to be a definition *(def)* in a transition, t, if x appears in the input portion of t or x appears at the left hand side of an assignment statement in the program segment of t. However, when a variable, x, appears on the right hand side of an assignment statement s of t or in an output statement s of t, x is both a computation-use *(c-use)* variable of t and s, respectively. If x is used in an enabling condition statement s of t, then it is a predicate-use *(p-use)* variable of t and s, respectively. In addition, p-use(t) denotes a set of p-use variables in t and c-use(t) denotes a set of c-use variables in t. Similarly, p-use(s) and c-use(s) are for s. A *global use* of x is a use of x whose *def* occurs in some other transitions; otherwise, it is a *local use*. Similarly, a *global use* of x is a def of x for which there is a global use in some other transitions; otherwise, it is a *local def*.

A *path* is a sequence of transition. A path is a *simple path* if all transitions, except possibly the first and the last, are distinct. A path is a *loop-free* path if all transitions are distinct. A path $(t_1, \cdots, t_n)$ is a *def-clear* path with respect to x if $t_2, \cdots, t_{n-1}$ do not contain def of x. A path $(t_1, \cdots, t_n)$ is a *du-path* with respect to a variable x if t_1 has a global def of x and either: 1) t_n has a global c-use of x and the path is a def-clear simple path with

Transition	def Variables	du-paths	Test Paths
1	cnt	1 → 3	(1,3,4)
		1 → 4	(1,4)
2	cnt	2 → 5 → 10	(1,2,5,10)
		2 → 5 → 8	(1,2,5,8)
		2 → 5 → 9	(1,2,5,9,10)
		2 → 5 → 7	(1,2,5,7,10)
	num	2 → 5, 2 → 5 → 6	(1,2,5,6,5,8)
		2 → 5 → 8	(1,2,5,8)
		2 → 5 → 9	(1,2,5,9,8)
		2 → 5 → 7	(1,2,5,7,8)
		2 → 5 → 9 → 7	(1,2,5,9,7,8)
		2 → 5 → 7 → 9	(1,2,5,7,9,8)
3	cnt	3 → 3	(1,3,3,4)
		3 → 4	(1,3,4)
5	data	5 → 9 → 7, 5 → 9	(1,2,5,9,7,14)
		5 → 7 → 9, 5 → 7	(1,2,5,7,9,14)
6	cnt	6 → 5 → 10	(1,2,5,6,5,10)
		6 → 5 → 8	(1,2,5,6,5,8)
		6 → 5 → 7	(1,2,5,6,5,7,10)
		6 → 5 → 9	(1,2,5,6,5,9,10)
	num	6 → 5 → 8, 6 → 5	(1,2,5,6,5,8)
		6 → 5 → 7	(1,2,5,6,5,7,8)
		6 → 5 → 9	(1,2,5,6,5,9,8)
		6 → 5 → 7 → 9	(1,2,5,6,5,7,9,8)
		6 → 5 → 9 → 7	(1,2,5,6,5,9,7,8)
7	cnt	7 → 7	(1,2,5,7,7,10)
		7 → 9	(1,2,5,7,9,10)
		7 → 8	(1,2,5,7,8)
		7 → 10	(1,2,5,7,10)
9	cnt	9 → 7	(1,2,5,9,7,10)
		9 → 9	(1,2,5,9,9,10)
		9 → 8	(1,2,5,9,8)
		9 → 10	(1,2,5,9,10)

Table 1: Test Paths for Testing Data Aspect of INRES

ETP No.	Executable Test Path	Corresponding Test Paths
1	$(1,3^4,4)$	(1,3,4), (1,4), (1,3,3,4)
2	$(1,2,5,7^4,10)$	(1,2,5,10),(1,2,5,7,10),(1,2,5,7,7,10)
3	$(1,2,5,7^4,8)$	(1,2,5,8),(1,2,5,7,8)
4	$(1,2,5,9^4,10)$	(1,2,5,9,10),(1,2,5,9,9,10)
5	$(1,2,5,9^4,8)$	(1,2,5,9,8)
6	$(1,2,5,6,5,7^4,10)$	(1,2,5,6,5,7,10),(1,2,5,6,5,10)
7	$(1,2,5,6,5,7^4,8)$	(1,2,5,6,5,8),(1,2,5,6,5,7,8)
8	$(1,2,5,6,5,9^4,10)$	(1,2,5,6,5,9,10)
9	$(1,2,5,6,5,9^4,8)$	(1,2,5,6,5,9,8)
10	$(1,2,5,7,9^3,10)$	(1,2,5,7,9,10)
11	$(1,2,5,7,9^3,8)$	(1,2,5,7,9,8)
12	$(1,2,5,9,7^3,10)$	(1,2,5,9,7,10)
13	$(1,2,5,9,7^3,8)$	(1,2,5,9,7,8)
14	$(1,2,5,6,5,7,9^3,8)$	(1,2,5,6,5,7,9,8)
15	$(1,2,5,6,5,9,7^3,8)$	(1,2,5,6,5,9,7,8)
16	(1,2,5,7,9,14)	(1,2,5,7,9,14)
17	(1,2,5,9,7,14)	(1,2,5,9,7,14)

Table 2: Executable Test Paths for Testing Data Aspect of INRES

respect to x or 2) t_n has a global p-use of x and the path is a def-clear loop-free path with respect to x. The all-du-paths criterion requires that all the du-paths for each variable in a given EFSM be covered by some paths in a test set. Therefore, selecting a set of paths to fulfill the all-du-paths criterion involves: 1) computing all the du-paths for each variable, and 2) finding a set of paths to cover these du-paths.

Table 1 shows all the du-paths extracted from the INRES protocol and a set of paths to cover these du-paths. Note that, du-paths that correspond to the same subsequence have only a single entry in this table. For instance, a subsequence 1 → 3 that is a du-path for p-use and a du-path for c-use of *cnt* has a common entry in the table. The main reason is that both the du-paths can be covered by the same test path. Each du-path d is covered by a path $(pre, d, post)$, where pre is the shortest path from the initial state of the protocol to the first transition of d and $post$ is the shortest path from the last transition of d to the initial state. Normally, not all paths in this set are executable due to the enabling conditions along the paths. In addition, since the dataflow technique is focused only on path selection, no clue is given to how path domains can be generated for developing test cases from these paths.

3 Effective Domain For Testing

In order to conform that a path chosen to cover some data interactions in a specification is indeed the same path taken by an IUT during execution, our method requires that every transition of the path be checked for control flow errors. This involves checking the tail state of every transition. Since the tail states cannot be observed directly, state check sequences must be appended to the respective transitions. However, a transition typically can exist in different test paths; therefore, if all the occurrences of a transition must be checked, then the length of the paths will be significantly increased. On the other hand, it is generally not sufficient to conclude that all occurrences of a transition are correct by simply testing one of these occurrences. It is due to the fact that an occurrence may be tested on some aspects of a transition that other occurrences cannot offer. (This point should become obvious after our discussion on effective domains.) In order to reduce the number of testing for a transition, strategies are needed for effectively selecting some occurrences of this transition so that fault coverage is not severely degraded. *Effective domain for testing of a transition in a path* provides a kind of measurement for evaluating the extent to which a transition can be tested in a given path.

The concept of effective domain for testing is based on the intuitive relationships between dataflow testing and domain testing [9]. The primary characteristic of domain testing is that a program's input domain is divided into subsets, with the tester selecting one or more elements from each subdomain. An *input domain* of a program is a *subset of* the Cartesian product of the respective range of values for each input variable. An element of an input domain is called an *input vector*. In a way, dataflow testing is a type of domain testing. Dataflow testing requires the exercising of path segments determined by combinations of variable definitions and variable uses. The input domain is divided so that there is a subdomain corresponding to each path covering the def-use associations [10]. A given subdomain contains every input vector that causes the particular def-use associations to be satisfied. This subdomain of a path is called the *path domain* of that path.

Let's call a tuple $< (x_1, v_1), \cdots, (x_n, v_n) >$, where $x_1, \cdots, x_n$ are all variables of a program and $v_i \in$ range of $x_i \cup \{U\}$, a *state vector* of the program. U is a null value. Then, a path domain defines an initial set of state vectors for a path. Based on this set, the execution of the first transition in turn determines the new set of state vectors for the second transition, and so forth. Each of these sets of state vectors before its corresponding transition is executed is collectively called the *executable state set* of that transition. Note that two transitions in a path may not be distinct transitions from a specification. Any possible execution of a transition (or rather an *occurrence* of a transition from a specification) in a path is restricted by its executable state set. Specifically, a tuple in this executable state set determines the respective values of those c-use and p-use variables of the transition. As a result, testing of this transition in the path can only be done on a certain combination of values of its c-use and p-use variables. With a given path, the allowed subsets of Cartesian product of the possible values for the p-use and c-use variables of a transition is collectively called the effective domain for testing of that transition in the path. The effective domain for testing provides a way that a tester can estimate how much a transition of a specification can be tested by its respective occurrences in a given path. Formally, the effective domain for testing is defined as follows:

Definition 1 *For a given path, $p = (t_{p_1}, \cdots, t_{p_n})$, the effective domain for testing of a transition, t_{p_i}, in p, denoted by $ED_{p,t_{p_i}}$, is*

$$\{proj_{x_{j_1}, \cdots, x_{j_m}}(e) \mid e \in \textit{execution state set of } t_{p_i}\}$$

where $x_{j_1} \in$ c-use$(t_{p_i}) \cup$ p-use(t_{p_i})
and $proj_{y_{j_1}, \cdots, y_{j_k}} < (y_1, v_1), \cdots, (y_p, v_q) > =$
$< (y_{j_1}, v_{j_1}), \cdots, (y_{j_k}, v_{j_k}) >$
for $1 \leq j_1 < j_2 < \cdots < j_k \leq q$

In the rest of this paper, if no ambiguity arises due to paths or transitions, an effective domain for testing of a transition in a path will simply be stated as an effective domain.

To illustrate the idea of effective domains, let's study a set of executable paths in Table 2, which is obtained from the test paths as shown in Table 1. Note that not all the test paths in Table 1 are executable. Some of the inexecutable paths can be made executable while computing their effective domains. The details will be presented in the next section. At this moment, let's assume that all executable paths have been obtained.

Table 3 shows the effective domains of the executable test paths (ETP) numbered 1, 2, 3, 5 and 7 in Table 2. For simplicity, $< (x_1, v_1), \cdots, x_i : \{a_1, \cdots, a_n\}, \cdots, (x_m, v_m) >$ is used to collectively denote the following state vectors $< (x_1, v_1), \cdots, (x_i, a_j), \cdots, (x_m, v_m) >$ for $1 \leq j \leq n$. Similarly, for a transition t_i (says, having two variables x and y) from Figure 1, $i < x : [v_1][v_2] \cdots [v_n], y : [u_1][u_2] \cdots [u_n] >$ is used to denote the n occurrences of t_i in a given path, where v_j and u_j are values of x and y, respectively, for the j^{th} occurrence of t_i. If y has the same value u in all n occurrences, then it will be denoted as $i < x : [v_1][v_2] \cdots [v_n], (y, u) >$. For the five paths, the effective domains for transition 1 are not shown. They are $< (cnt, U) >$. In ETP 1, each occurrence of transition 3 can be tested with $cnt = 0$, $cnt = 1$, $cnt = 2$ and $cnt = 3$, respectively. This is because the first time transition 3 is tested, the value of cnt has been set to 0 by transition 1. Then, cnt is increased by one by the statement $cnt := cnt + 1$. As a result, the second time transition 3 is tested, the transition is tested with the value of cnt equal to 1. The same argument applies to the third and forth iterations of transition 3.

In Table 2, it also can be seen that transition 8 has different effective domains in ETP 3 and ETP 7. ETP 3 has $< (cnt, 4), (num, 1), N : [2..] >$, but ETP 7 has $< (cnt, 4), (num, 2), N : [3..] >$. However, under many circumstances, a transition that appears in two different paths can have the same effective domain. For instance, transition 8 has the same effective domain of $< (cnt, 4), (num, 1), N : [2..] >$ in both ETP 3 and ETP 5, and transition 9 has the same effective domain of $< cnt : [0][1][2][3], (num, 1), data : [a..z] >$ for ETP 5 and ETP 7. Therefore, ETP 3 and ETP 5 can only provide the same effectiveness in terms of ranges of values for testing transition 8, and ETP 5 and ETP 7 can only provide the same for transition 9.

Thus, using effective domains, the occurrences of a transition in different paths can be qualitatively evaluated. For a transition, our strategy is then to choose a subset of occurrences that can adequately cover all the

ETP No.	Executable Test Path	Effective Domain For Testing
1	$1,3^4,4$	$3 < cnt : [0][1][2][3] >$ $4 < (cnt, 4) >$
2	$1,2,5,7^4,10$	$2 < (cnt, 0) >$ $5 < (num, 1), data : [a..z] >$ $7 < cnt : [0][1][2][3], (num, 1), N : [2..), data : [a..z] >$ $10 < (cnt, 4) >$
3	$1,2,5,7^4,8$	$2 < (cnt, 0) >$ $5 < (num, 1), data : [a..z] >$ $7 < cnt : [0][1][2][3], (num, 1), N : [2..), data : [a..z] >$ $8 < (cnt, 4), (num, 1), N : [2..) >$
5	$1,2,5,9^4,8$	$2 < (cnt, 0) >$ $5 < (num, 1), data : [a..z] >$ $9 < cnt : [0][1][2][3], (num, 1), data : [a..z] >$ $8 < (cnt, 4), (num, 1), N : [2..) >$
7	$1,2,5,6,5,9^4,8$	$2 (cnt, 0) >$ $5 < num : [1][2], data : [a..z] >$ $6 < (num, 1), (N, 1) >$ $9 < cnt : [0][1][2][3], (num, 2), data : [a..z] >$ $8 < (cnt, 4), (num, 2), N : 1 or [3..) >$

Table 3: Effective Domain of Some Test Paths

distinct effective domains. Since the reasoning for effective domains is parallel to that of domain testing, the commonly used definition for distinct domains in the context of domain testing is adopted in our context; that is, two domains are said to be distinct if they do not have the same set of variables of which each variable has an identical domain. In this sense, two domains such that one is a subset of the other are still considered distinct.

4 Test Path Selection

Assuming that a set of paths, which may contain inexecutable paths, has been selected for fulfilling the all-du-paths criterion, our test path selection algorithm will perform two more tasks. They are computing the effective domains for the paths and selecting the occurrences of the transitions.

4.1 Computing Effective Domains

Since not all the transitions in a specification are included in the set of selected paths, those yet-to-be selected transitions need to be included in this set. For this purpose, a tour containing these transitions will be added to the set.

Our strategy in computing effective domains for a given path is to progressively computing the executable state sets for a newly encountered transition while traversing the path and modifying the executable state sets of some traversed transitions to reflect the constraints imposed by the newly visited transition. Each of the conditional and assignment statements imposes a certain constraint on those variables that appear in the statement. The allowed values for a variable will be represented by the allowed maximum and the allowed minimum values of the variable, and a list of constraints that specifies the relations of this variable with some other variables.

Initially, the executable state set of the initial transition consists of the Cartesian product of the variables. When a transition is encountered, each of its enabling condition t will be evaluated against the current boundaries of the involved variables. This evaluation is done by computing the new boundary of a selected variable v among c-use(t). The selection prefers input variables to context variables. This preference is to reduce the chances of failure and the number of changes required for the executable state sets along the path. Based on this computed boundary D, the logic for further execution is as follows:

```
If $D$ is not empty then
 If the new constraint conflicts with existing constraints
  then path may be inexecutable
 else
  If $D$ is a subset of the old domain $D'$ of $v$
   then do nothing
  else
   $D := D \cap D'$
   propagate $D$ to the executable state sets of which
   the corresponding transitions have variables directly or indirectly dependent on $v$ and update the
   constraints of these transitions
else path may be inexecutable
```

For checking the consistency of the new constraint against the existing constraints, only those constraints directly or indirectly related to those variable in c-use(t) need to be examined. Generally, the constraints for communicating protocols are in simple forms, such as linear form, and a list of the relavent constraints are normally very short. For instance, at most of the time, the constraint list for most of the variables in INRES has only a single constraint. In addition, since the domains of the variables are known, consistency of the relevant constraints can be easily checked by the constraint satisfaction problem (CSP) technique [11].

In the context of communicating protocols, most of the inexecutable paths can be made executable if some sequences of transitions are inserted into the original paths. This always involves inserting a cycle to a particular state of an original path to change the values of v. A loop that can alter the values of v will be chosen. The number of times this loop must be unfolded is determined by the number of execution needed to bring the values of v within D. This loop will be unfolded a number of times as required and inserted into the given test path. The constraints of the related transitions will be updated accordingly.

An assignment statement is not only to define the value of its def variable, but also to impose constraints on the def and c-use variables of the statement. This constraint will be used to restrict some c-use variables if the possible values of the def variable has to be restricted based on some constraints imposed by later statements. An output statement does not impose any constraint on the variables involved.

Based on an executable state set, the corresponding effective domain can be computed as the boundaries of the p-use and c-use variables, and constraints relevant to these variables. It should be noted that for most of the test sequence generation methods proposed in the literature, the test path selection and test case selection are separated. For those test sequence generation methods based on dataflow criteria, a step which may be manual walkthrough or computed automatically by some symbolic execution methods, is unavoidable for computing the path domains and choosing test cases based on the paths in the given test set. In our method, the path domain as well as the constraints on the input variables are computed. These two pieces of information will help in generating input vectors.

4.2 Selecting Occurrences of a Transition

After computing effective domains for the occurrences of each transition, the next task is to select those occurrences that have distinct effective domains for a transition. Identical effective domains of a transition are its occurrences that have identical boundaries and constraints for the relevant variables. However, some care must be taken to reduce the overall length of the resulted test paths. There are two reduction rules that can be employed to reduce the number of occurrences of a transition before making selection. The first rule is to be applied to a set of occurrences that have distinct domains, and the second rule is to be applied to a set of occurrences that have identical effective domains. The first reduction rule is as follows:

R1 **Redundant Domains**: An effective domain, $ED_{p_j,t}$ can be eliminated if there exists some $ED_{p_i,t}$ (says n) for $1 \leq i \leq n$ and $i \neq j$ such that

$$ED_{p_j,t} = \cup_{i=1}^{n} ED_{p_i,t}$$

where $\cup$ is defined for a tuple of range of values as follows:

$$\cup_{i=1}^{n} < D_{i1}, \cdots, D_{im} >=< \cup_{i=1}^{n} D_{i1}, \cdots, \cup_{i=1}^{n} D_{im} >$$

In R1, the effective domain $ED_{p_j,t}$ is made redundant by the n effective domains $ED_{p_i,t}$ since testing of t in p_j with any possible values for variables of t before t is executed can also be conducted with some p_i.

Before describing the second reduction rule, let's discuss the problems of appending state check sequences to transitions. For each transition, state check sequences must be appended to the tail states of those selected occurrences. To ensure that the fault coverage of these paths is not severely affected after state check sequences are introduced, these newly added state check sequences must be selected with care. In general, introducing more transitions to a path of an FSM cannot degrade the fault coverage of that path, but introducing more transitions in a path of an EFSM may degrade the fault coverage of that path. A simple example is that a newly introduced segment may redefine some variables in a path that is to cover a def-clear segment. Three types of state check sequences can be used to serve the purpose of identifying a tail state. The preference that it is chosen for a tail state is according to the following order.

Type 1 An unique check sequence such as an UIO sequence for a state such that this sequence overlaps with the subsequent transitions.

Type 2 A sequence that concatenates 1) a unique check sequence for a state with 2) a transfer sequence to bring the flow back to the tail state, and does not redefine the def variables of any def-use association that the original path is supposed to cover.

Type 3 A sequence that concatenates a unique check sequence for a state with a transfer sequence to bring the flow back to the initial state and then back to the tail state.

Obviously, state check sequences of Type 1 do not alter the effective domains of transitions in a path. In addition,

States	UIO Sequences
DISCONNECTED	1
WAIT	3 or 2
CONNECTED	5
SENDING	7 or 6 or 8 or 9

Table 4: State Check Sequences for INRES

Type 1 sequences do not increase the length of a path. Type 3 requires that all the variables be reset after a state check sequence is applied. It is, thus, less preferred than the other two types of check sequences.

The second reduction rule is based on the Type 1 state check sequences. It deals with a set of occurrences of a transition that have an identical effective domain. If there exists an occurrence in this set that has an UIO of the transition overlapping with its path, this occurrence can be chosen as the representative for this set of occurrences. However, since this occurrence has an UIO of the transition overlapping with the path, no extra checking is needed at all. Thus, the second rule can be formulated as follows:

R2 **Overlapping Transitions**: No checking is needed for a set of occurrences of a transition t that has an identical effective domain if there exists a path p_i in this set such that p_i contains a subsequence (t, U), where U is a state check sequence for the tail state of t.

An application of this rule to some paths shown in Table 2 is as follows: As shown in Table 3, both ETP 2 and ETP 3 have the same effective domain of $< (cnt, 3), (num, 1), N : [2..), data : [a..z] >$ for the forth iteration of transition 7. By applying R2, the forth iteration of transition 7 in ETP 2 needs not be checked. It is because ETP 3 contains a subsequence (7,8). As shown in Table 4, transition 8 is an UIO of the tail state of transition 7.

Some caution of words must be given here. By applying R1 to a set of test paths, the fault coverage for the resulting set cannot be degraded. However, applying R2 may degrade the fault coverage of a set. According to [12], the fault coverage of UIOs for a set of transitions can be degraded if these UIOs are overlapped with the subsequent transitions in a test path that is supposed to test this set of transitions. Since the tail state of a transition is always tested a number of times by our method in different occurrences, the chances of degrading are reduced. Nevertheless, Miller and Paul [13] pointed out that if there is no *converging state* in a path, fault coverage is not compromised by overlapping the UIOs of the tail states of the intermediate transitions. Converging states are those states which have outgoing transitions with identical input/output that go to the same state.

ETP No.	Final Test Path
1	$(1,3^4,\underline{2,13,1,3^4},4,\underline{1,12})$
2	$(1,2,5,7^4,10,\underline{1,12})$
3	$(1,2,5,7^4,8,\underline{1,12})$
4	$(1,2,5,9^4,10)$
5	$(1,2,5,9^4,8)$
6	$(1,2,5,6,5,7^4,10)$
7	$(1,2,5,6,5,7^4,8,\underline{1,12})$
8	$(1,2,5,6,5,9^4,10)$
9	$(1,2,5,6,5,9^4,8)$
10	$(1,2,5,7,9^3,10)$
11	$(1,2,5,7,9^3,8)$
12	$(1,2,5,9,7^3,10)$
13	$(1,2,5,9,7^3,8)$
14	$(1,2,5,6,5,7,9^3,8)$
15	$(1,2,5,6,5,9,7^3,8)$
16	$(1,2,5,7,9,14,\underline{1,12})$
17	$(1,2,5,9,7,14)$
18	$(1,3,12,\underline{1,12},11,1,2,13,\underline{1,12})$

Table 5: A Final Set of Test Paths for INRES

After applying these two rules, it will be easy to decide for which the occurrences of a transition, state check sequences are needed. For those domains that have only one occurrence, these occurrences will be checked. For each of those domains that have more than one occurrences, only one is needed. Of all these selected occurrences, Type 2 or Type 3 sequences can be used for each of them.

4.3 An Example

Based on the aforementioned algorithm, a final test set that can test both the data flow and the control flow of INRES is given in Table 5. The state check sequences are underlined. Except that the state check sequences for transition 3 in ETP 1 are for the WAIT state, the others are all for the DISCONNECTED state. Even though transition 8 appears 8 times, the tail state of transition 8 is only tested at ETP 3 and ETP 7. Similarly, the tail state of transition 10 is tested once at test sequence 2 even though transition 10 appears 6 times. Indeed, transitions 8 and 10 use the same state check sequence; it is $(1, 12)$ and is of Type 2. Transition 1 is an UIO of DISCONNECTED and transition 12 is a transfer sequence to bring the flow back to DISCONNECTED. This same state sequence is also used for testing transition 13 in ETP 13 and transition 14 in ETP 16.

In ETP 1, the last execution of transition 3 that has an effective domain of $< (cnt, 3) >$ is tested by a Type 3 state check sequence. This sequence consists of transition 2 and a subsequence, $(13,1,3^4)$. Note that the correctness of transition 3 is checked at the first three iterations by a Type 1 state check sequence which is (3). Since transition 3 is tested in ETP 1 with an effective domain of $< (cnt, U) >$, it is not tested in ETP 18 again.

5 Conclusion

In this paper, an approach is proposed for generating paths that check both data and control flow of a protocol. After generating a set of paths to cover a dataflow selection criterion, our method uses the concept of effective domains to selectively augment the transitions in the set with state check sequences so that the control flow can be ensured and the dataflow coverage can still be preserved. Our method also computes path domains for the paths and makes some inexecutable paths executable.

In terms of fault detection capability, our method preserves the fault coverage of the dataflow analysis technique since the def-use associations of a criterion that a selected path is supposed to cover are not disturbed by any of the three types of state check sequences. In addition, our approach also ensures that every transition and its tail state is tested at least once. This is the minimum requirement in the conventional sense for checking control errors. However, as mentioned in Section 3, it is not sufficient for the EFSM. The resulting test sets further ensure that every distinct effective domain is covered. For a transition, our algorithm only appends state check sequences to those occurrences that have distinct effective domains, thereby reducing the number of state check sequences to be appended to a test set.

To the best of our knowledge, there are only two works [14, 15] that have dealt with the issue of combining control flow testing with dataflow testing. But, neither has proposed ways for computing path domains for the respective paths. In [14], a combining method is used for generating a test path for a given mutant. Basically, this method uses segment overlapping technique to shorten the overall length of a test path and only requires that a transition be tested once. As mentioned in Section 3, a transition in an EFSM generally must be tested more than once for revealing errors that may only occur in certain paths. In [15], a method is proposed to augment a set of paths generated to cover the all-du-paths criterion with cyclic characterizing sequences (CCS). A CCS is the same as a Type 2 state check sequence. By applying Miller's result reported in [13] for the FSM, this method only inserts a CCS to the last state of a path if there is no converging state along the path. In the case that several converging states appear in a path, each converging state is replaced by a CCS. In our method, these two cases are taken care of by the Type 1 and Type 2 sequences. In addition, our method do not check a transition whose tail state is a converging state more than once if its occurrences have the same effective domain. In our example, if transition 7 is not an UIO for SENDING, then our method will only check transition 7 for its four iterations in ETP 2. However, the method of [15] will have to check all occurrences of transition 7 in various paths.

Regarding our future work, strategies are being investigated for using the underlying control structures of a specification to reduce the number of test cases needed for data flow testing.

References

[1] K. Sabnani and A. Dahbura, "A protocol test generation procedure," *Computer Networks and ISDN Systems*, vol. 15, pp. 285–297, 1988.

[2] T. Chow, "Testing software design modeled by finite-state machines," *IEEE Trans. on Software Engineering*, vol. SE-4, no. 3, pp. 178–187, March 1978.

[3] B. Sarikaya, G. v. Bochmann, and E. Cerny, "A test design methodology for protocol testing," *IEEE Trans. on Software Engineering*, vol. SE-13, no. 5, pp. 518–531, May 1987.

[4] H. Ural, "Test sequence selection based on static data flow analysis," *Computer Communications*, vol. 10, no. 5, pp. 234–242, 1987.

[5] H. Ural and B. Yang, "A test sequence selection method for protocol testing," *IEEE Trans. on Communications*, vol. 39, no. 4, pp. 514–523, April 1991.

[6] D. Hogrefe, "OSI formal specification case study: The INRES protocol and service," tech. rep., IAM 91-012, University of Berne, Institute of Computer Science and Applied Mathematics, 1991.

[7] S. Rapps and E. J. Weyuker, "Selecting software test data using data flow information," *IEEE Trans. on Software Engineering*, vol. 11, no. 4, pp. 367–375, April 1985.

[8] H. Ural and B. Yang, "A structural test selection criterion," *Infomation Processing Letter*, vol. 28, no. 3, pp. 157–163, 1988.

[9] L. J. White and E. I. Cohen, "A domain strategy for computer program testing," *IEEE Trans. on Software Engineering*, vol. 6, no. 3, pp. 247–257, May 1980.

[10] E. J. Weyuker and B. Jeng, "Analyzing partition testing strategies," *IEEE Trans. on Software Engineering*, vol. 17, no. 7, pp. 703–711, July 1991.

[11] A. K. Mackworth, "Consistency in networks of relations," *Artificial Intelligence*, vol. 8, no. 1, pp. 99–118, 1977.

[12] M.-S. Chen, Y. Choi, and A. Kershenbaum, "Approaches utilizing segment overlap to minimize test sequences," in *Proc. of 10th IFIP Symp. on Protocol Specification, Testing, and Verification*, pp. 67–84, 1990.

[13] R. E. Miller and S. Paul, "Generating minimal length test sequences for conformance testing of communication protocols," in *Proc. IEEE INFOCOM '91*, pp. 970–979, 1991.

[14] R. E. Miller and S. Paul, "Generating conformance test sequences for combined control and data of communication protocols," in *IFIP Trans. Protocol Specification, Testing, and Verification, XII*, pp. 1–15, North-Holland, 1992.

[15] S. T. Chanson and J. Zhu, "A unified approach to protocol test sequence generation," in *Proc. IEEE INFOCOM '93*, pp. 106–114, March 1993.

F. C. J. Wang, L. S. Koh and M. T. Liu,
"Protocol Validation Tools as Test Case Generators,"
Proc. 7th IFIP Int'l Workshop on Protocol Test Systems,
pp. 149–164, November 1994.

Protocol Validation Tools as Test Case Generators*

Chang-Jia Wang, Liang-Seng Koh and Ming T. Liu

Department of Computer and Information Science, The Ohio State University,
2036 Neil Ave., Columbus, Ohio 43210-1277, USA

Abstract
A method is proposed to transform test case generation problem to protocol validation problem. Protocol validation has been studied for years and many validation tools are available. By transforming a test case generation problem into a protocol validation problem, a protocol validation tool can be used to generate test cases. The method can be implemented in a very short period of time. The complexity of the proposed method in searching for a test case is $O(n)$, where n is the number of system states in the specification.

1 Introduction

Test case generation (TCG) for *Finite State Machines (FSM)* has been studied for years, and fruitful results have been produced [1, 2, 3, 4]. The FSM is a simple model that ignores the data flow of a protocol and concentrates on the protocol's control flow. In other words, it simplifies a protocol into a state-transition machine. However, ignoring the data flow of the protocol often removes many interesting properties from the protocol. As an example, it is hard to specify a simple protocol like Go-Back-N in FSM without losing those important properties associated with the protocol's window size.

Therefore, most protocol specifications are written in more sophisticated models, such as *Extended Finite State Machine (EFSM)* [5], *Estelle* [6], *LOTOS* [7], or other programming languages. These models are called *extended models* in this paper with respect to the basic FSM model. To be more practical, a test case generation method must be able to generate test cases from extended models. However, the difficulty is that to test an extended model, one needs to test not only the control flow, but also the data flow of the protocol. Testing both control and data flows is considered a challenging problem and

*Research reported herein was supported by U.S. Army Research Office, under contracts No. DAAL03-91-G-0093 and No. DAAL03-92-G-0184. The views, opinions, and/or findings contained in this paper are those of the authors and should not be construed as an official Department of the Army position, policy or decision.

not much work has been done so far. Some references about the test case generation for the extended models can be found in [8, 9, 10, 11, 12, 13, 14].

A test case can be used for detecting a certain type of faults, which are called the *fault models* of the test case [15]. Because of the complexity involved in searching test cases for extended models, it is believed that *fault models* should be used as a guide to simplify the search [16, 17]. Traditionally, a TCG method is dedicated to generating test cases for certain fixed fault models. On the contrary, a TCG method based on fault models takes both a protocol specification and a given fault model as input and generates a test case for the fault model. An I/O sequence can be used as a test case if the specification and the fault model behave differently when applying the I/O sequence.

A fault model is usually a modification of the original protocol specification, and can be described by the same model used to define the protocol specification. For example, Figure 1 is the sender of a Go-Back-N protocol specified in EFSM with a window size equal to four (W=4). A fault model like the one drawn in Figure 2 has a window size being set to five. Note that only the action associated with the first transition is changed[†].

Generating test cases based on a given fault model has the following advantages:

1. Every test case has a well defined *test purpose*. That is, the types of errors a test case can detect is automatically provided. In addition, those who test a protocol implementation with the test case will have the confidence that the fault described by the fault model does not exist in the implementation.

2. It reduces the complexity of finding a test case. Instead of searching aimlessly, a TCG method can use a given fault model as a goal to direct the search.

3. Users can chose to test critical faults first, and test minor errors later.

4. It is more flexible than the traditional methods. The traditional TCG methods have fixed fault models embedded in them. If a critical fault does not contain in the fault models, the traditional TCG methods cannot effectively test this fault.

[†] The EFSMs used in this paper have five types of actions associated with their transitions.

1. Assignment: $W := 4$ assigns value 4 to variable W.
2. Input: $R?Ack(A)$ receives message Ack whose argument is assigned to variable A.
3. Output: $R!Msg(B[Sm], Sm)$ sends a message Msg with arguments $B[Sm]$ and Sm, where $B[Sm]$ is the Sm-th element in array B.
4. Condition: A transition with action $k <> Sm$ can only be executed when k and Sm are not equal.
5. Timeout (T/O): The transition will be executed after a certain amount of time expires.

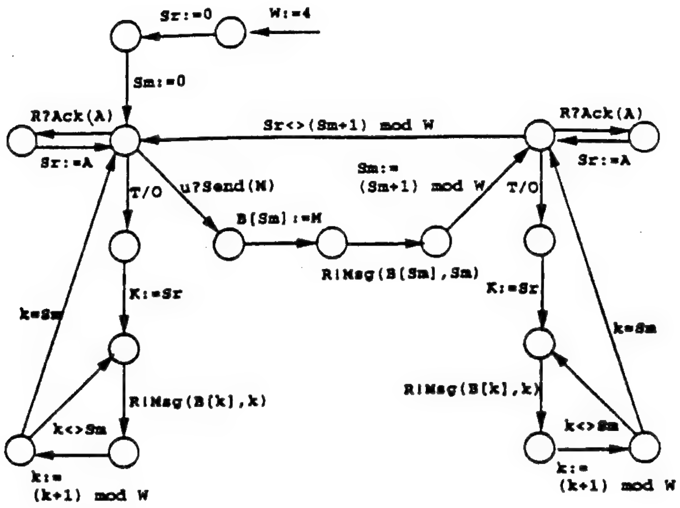


Figure 1: Go-Back-N protocol specified in EFSM

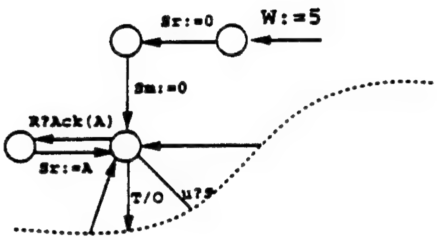


Figure 2: A fault model of Figure 1 (The rest of this figure is the same as Figure 1

In this paper, the problem of TCG based on fault models is transformed into a *protocol validation (PV)* problem. Protocol validation, which is used to detect design errors in a protocol specification, has been studied for years. While many protocol validation tools have been made available, automatic test case generators are still hard to find. Therefore, a method is proposed to transform TCG to PV so that available validation tools can be used to generate test cases. It will also be shown later that the transformation method not only can be used to generate test cases for weak conformance, but also can be used to generate test cases for strong conformance, functional testing, and multiple-module specifications.

It is well known that there is a so called *state explosion* problem for protocol validation, which means that the number of states generated by a protocol validation tool increases exponentially when the protocol specification becomes more complicated. The method proposed in this paper takes a protocol specification and a fault model as its input and generates a combined system to be fed into a protocol validation tool. It is also shown in this paper that the complexity of the combined system is $O(n)$ if the number of global states in the original protocol specification is n. Therefore, the state explosion problem will not occur as long as it will not occur in the original specification.

The organization of this paper is as follows. In Section 2, the transformation from the test case generation problem to the protocol validation problem is proposed. The same section also discusses how the transformation to weak and strong conformance, functional testing, and multiple-module protocols can be applied. In Section 3, the implementation of the test case generator is introduced, and some experimental results are discussed. Finally, the complexity of the test case generator is discussed in Section 4 and a conclusion is given in Section 5.

2 Transforming TCG to PV

2.1 The Transformation

A test case for a fault model is an I/O sequence that detects the discrepancies between the fault model and the protocol specification. That is, the test case is a possible I/O sequence for one, but not for the other. For example, an I/O sequence can be used as a test case if it can be generated from the protocol specification, but cannot be reproduced from the fault model. In other words, if a protocol implementation contains an error described by the fault model, applying the input sequence of the test case to the implementation will produce an unexpected output because the I/O sequence cannot appear in the implementation.

The task is to find an I/O sequence that can only be applied to either the protocol specification or the fault model, but not both. Let P (for *Primary*) be an automaton that describes either the protocol specification or the fault model, and let S (for *Secondary*) be an automaton that describes the other. If P and S are executed synchronously, whenever P makes a certain output, S must make the same output; otherwise, it will be a discrepancy. Similarly, if P requires a certain input, S must wait for the same input; otherwise, it will be a discrepancy as well.

The idea here is to construct a system that will deadlock when P and S take different input or output actions. The combined system can be fed into a protocol validation tool. When the validation tool signals a deadlock error, one can trace through the execution path that causes the deadlock. Then, the I/O events generated by the path can be used as a test case.

The first step is to modify automaton P. Let P' be an automaton the same as P, except for the following differences:

1. For every output action in P, make the same output. Then, wait for a `resume` signal.
2. For every input action in P, output a message that requests the input first. Then, wait for the input.
3. For every timeout in P, output a message labeled `T/O`. Then wait for a `resume` signal.

Next, make an automaton S' similar to S except for the following changes:

1. For every output action in S, input a message first. If the input message is the same as the message to be output, output an ok signal.
2. For every input action in S, input the message. Then, output an ok signal.
3. Remove every timeout in S.

Finally, let M (for *Monitor*) be an automaton communicating with P' and S'. M provides a variable for each parameter of every input action in P'. The variables are initialized to their lower bound values. Monitor M performs the following:

1. Increment the value of a variable by one nondeterministically, unless an upper bound is reached.
2. When an output from P' is received, pass the message to S' and wait for an ok signal from S'. After receiving the ok signal, send a `resume` signal to P', and record the output event.

Table 1: Modification made by P', S', and M

	Original	Primary P'	Secondary S'	Monitor M
Output	$G!msg(x_1, \cdots, x_n)$	$M!msg(x_1, \cdots, x_n)$ $M?resume$	$M?msg(p_1, \cdots, p_n)$ $p_1 = x_1$ $\vdots$ $p_n = x_n$ $M!ok$	$P'?msg(p_1, \cdots, p_n)$ $S'!msg(p_1, \cdots, p_n)$ $S'?ok$ $P'!resume$
Input	$G?msg(x_1, \cdots, x_n)$	$M!request(msg)$ $M?msg(x_1, \cdots, x_n)$	$M?msg(x_1, \cdots, x_n)$ $M!ok$	$P'?request(msg)$ $S'!msg(x_1, \cdots, x_n)$ $S'?ok$ $P'!msg(x_1, \cdots, x_n)$
Timeout	T/O	$M!T/O$ $M?resume$		$P'?T/O$ $P'!resume$

3. When P' asks for an input message, send the message to S' first. After receiving a ok from S', send the same message to P' and record the input event.

4. When P' outputs a T/O message, send **resume** signals to P', and record the timeou event.

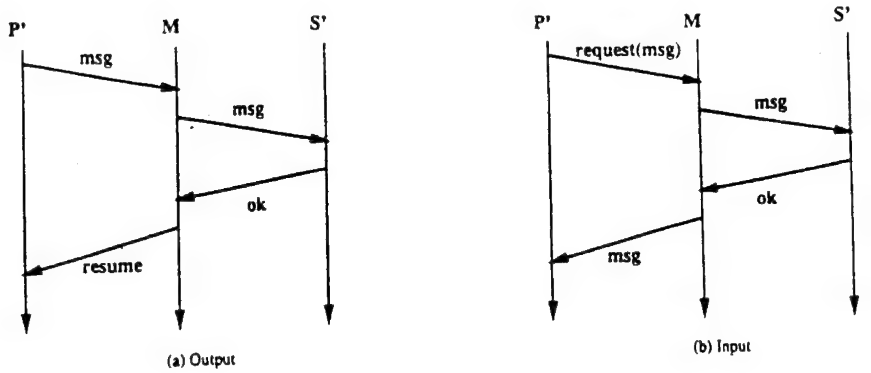
The above modification is summarized in Table 1. Let $\mathcal{I}$ (for *Integrated*) be an integrated system that contains P', S', and M. Then, $\mathcal{I}$ will deadlock if P generates som I/O sequences that cannot be reproduced by S. This can be briefly proved as follows.

1. If S cannot generate the same message output by P, S' will stop. Without receiving any **ok** signal, M will be blocked and will not send the **resume** signal to P'. Therefore, P' will be blocked as well.

2. If S will not receive the same input as P does, S' will be blocked and no **ok** signal will be send. Therefore, M will be blocked and P' will never receive the **resume** signal. The system deadlocks.

The protocol among M, P' and S' is illustrated in Figure 3. When P' sends a message **msg** to M, M passes it to S'. If the **msg** is also a message that S' intends to output, S' will send an **ok**. Upon receiving the **ok** signal, M sends a **resume** to P' (Figure 3a). On the other hand, if P' needs to receive a certain message, it sends a request to M. Upon receiving the request, M sends the message to S' first. If an **ok** is received, M sends the requested message to P' (Figure 3b). The timeout message between P' and M' are used to record the timeout event, and will not affect the overall execution of the system.

An example of the transformation is shown in Figures 4 to 6. Figure 4 is the primary automaton P' modified from the Go-Back-N protocol specification in Figure 1. The



(a) Output

(b) Input

Figure 3: Communication among M, P' and S'

shaded arrows and circles are the transitions and states being changed according to the rules described in Table 1. Figure 5 is the secondary automaton S' modified from the fault model shown in Figure 2. Monitor M for Figures 4 and 5 is drawn in Figure 6. Note that all the I/O events in P' and S' are changed into communication between P' and M, and between S' and M, respectively.

When a protocol specification and its fault models are specified by a formal model (such as FSM, EFSM, Estelle, LOTOS, SDL, Petri-Net, Promela, etc.), it is trivial to transform P and S into P' and S', respectively. It is also straightforward to construct the monitor M. Automata P', S' and M can be fed into a validation tool for the formal model and a deadlock can be found if there is an I/O sequence that is possible for P but cannot be generated by S. By retracing the path leading to the deadlock, monitor M will record the I/O events along the path. The recorded I/O event sequence can then be used as a test case.

2.2 Weak and Strong Conformance

There are two types of conformance between a protocol specification and its implementation. The *weak conformance* indicates that a protocol implementation should perform every function defined in its specification. In other words, things that ought to happen should happen. The *strong conformance* imposes an additional constrain which states that a protocol implementation cannot have any behavior not defined by the protocol specification. That is, things that cannot happen must not occur.

A fault model violating the weak conformance cannot perform some functions defined by the protocol specification. In other words, there exists an I/O event sequence that

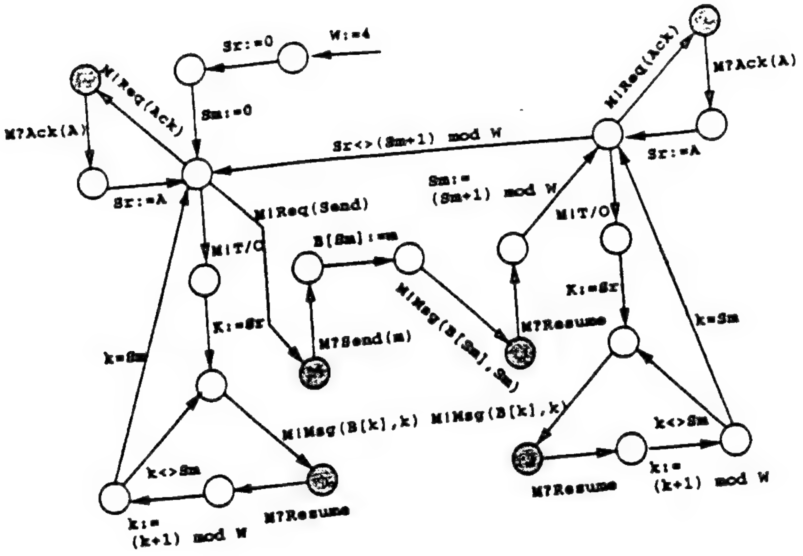


Figure 4: Modified protocol specification P'

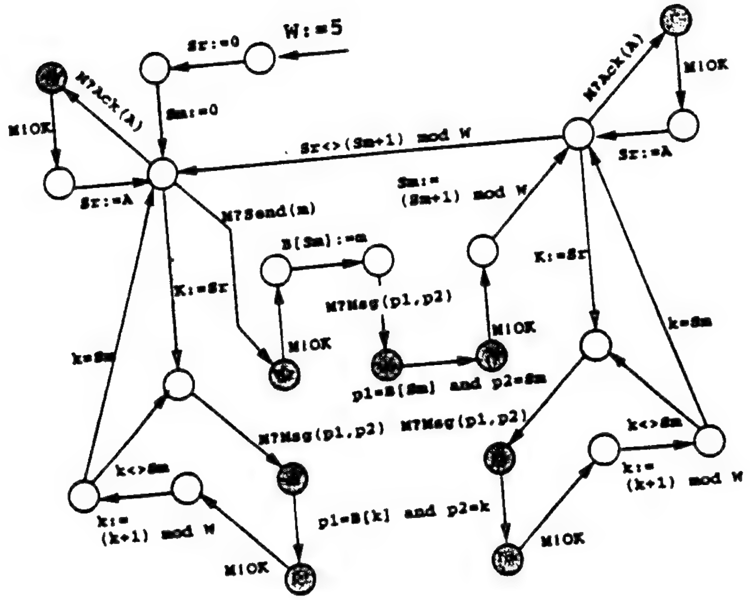


Figure 5: Modified fault model S'

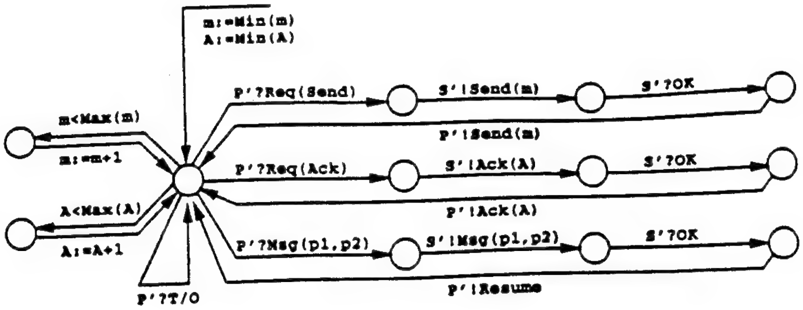


Figure 6: Monitor M

can be generated by the specification, but cannot happen in the fault model. To generate a test case for the fault model, one can use the protocol specification as the primary automaton P, use the fault model as the secondary automaton S, apply the transformation in Section 2.1, and feed the resulting system $\mathcal{I}$ into a protocol validation tool.

A fault model that violates the strong conformance has some execution paths that are not specified in the protocol specification. That is, some I/O sequence generated by the fault model can never happen in the specification. Thus, if one use the fault model as the primary automaton P and use the specification as the secondary automaton S, one can find a test case with the method described in Section 2.1. When the input of the test case is applied to a protocol implementation, and an expected output appears, the implementation must contain an error described by the fault model.

It is possible that a fault model violates both weak and strong conformance. In this case, treating the protocol specification as either the primary or the secondary will work. Sometimes, however, it is difficult to tell whether a fault model violates the weak or strong conformance. In such a case, if one method cannot find a test case, the other must be tried.

2.3 Functional Testing

It is useful to create test cases that ensure the correctness of certain behavior in the protocol specification. Such "behavior" is often called a *function* or a *requirement* of the protocol specification. For example, a requirement of the Go-Back-N protocol in Figure 1 can be "*there are no more than three messages in the channel at any given time.*"

In order to test if a protocol implementation conforms with the requirement, a fault model that violates the requirement is constructed. However, since a requirement usually is just a small portion of the original protocol specification, it will be quite troublesome to construct a fault model from the entire protocol specification. In fact, a fault model can be created by including only those transitions necessary to violate the requirement. For example, a fault model violating the requirement described in the above paragraph can be drawn as an EFSM in Figure 7. In the figure, there is no restriction on how many messages can be sent to the receiver consecutively before the sender receives an acknowledgment.

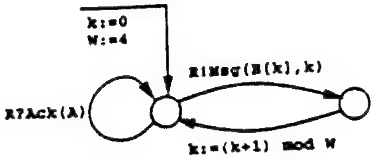


Figure 7: A fault model that sends more than three messages at a time.

The fault model in Figure 7 defines some behavior that should not exist in a correct implementation, so it violates the strong conformance requirement. Therefore, it is used as a primary automaton and the specification is used as a secondary. Since the fault model only specifies partial behavior of the entire specification, some I/O events in the specification may not exist in the fault model. Hence, a modification of the transformation described in Section 2.1 should be noted as follows.

- To change P (S) into P' (S'):
  1. If an input or output action exists in both P and S, follow the procedure in Section 2.1.
  2. For every output message that exists only in P (S), send the output to M and wait for a **resume** signal from M.
  3. For every input message that exists only in P (S), send an request for the input to M and wait for the input event from M.
- To construct M:
  1. For those input or output actions that exist in both P and S, follow the procedure in Section 2.1.
  2. If an output message that exists only in P (S) is received from P' (S'), record the output event and send a **resume** signal to P' (S').
  3. If a request for input that exists only in P (S) is received from P' (S'), record the input event and send the input message to P' (S').

In other words, if an input or output message exists only in P or S, the monitor M will not try to synchronize P' and S' with the message. It will only record the I/O event and release the automaton that generates the event. For example, in Figure 1, the input event u?Send(m) exists only in the specification, and should be modified according to the above procedure. The resulting EFSMs P', S', and M are shown in Figures 8, 9, and 10, respectively. Note that the shaded nodes and arrows in Figures 9 and 1 show the difference between the method in this section and the one in Section 2.1.

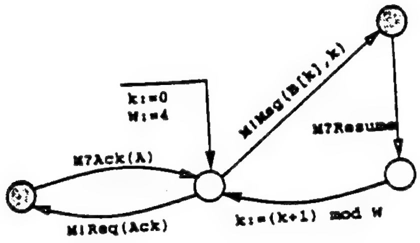


Figure 8: The primary P' from the fault model in Figure 7.

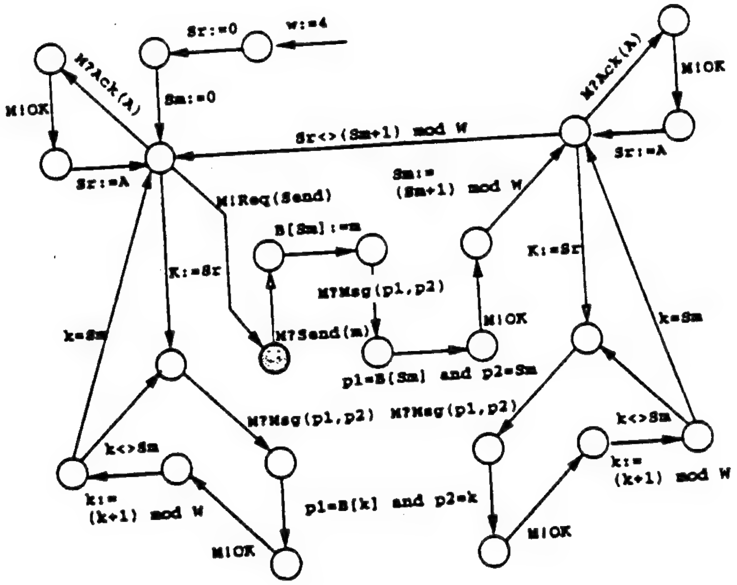


Figure 9: The secondary S' from the specification in Figure 1.

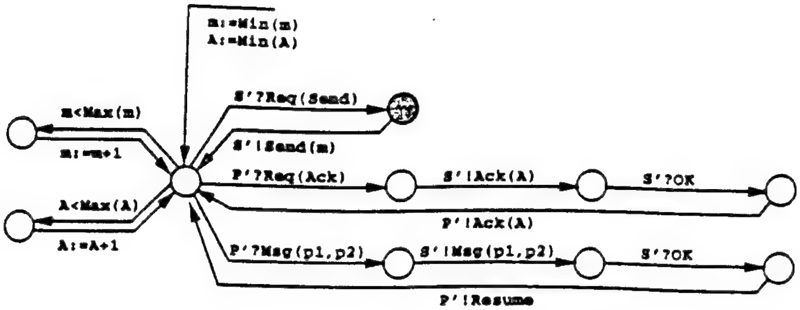


Figure 10: The monitor M from the fault model in Figure 7 and the specification in Figure 1.

2.4 Test Case Generation for Multiple Modules

Test cases can also be generated for multiple modules using this method. For example, one may want to test how a protocol containing a sender and a receiver operates when connected with a network. The whole system can be modeled like the one in Figure 11.

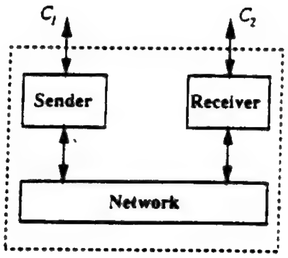


Figure 11: A multiple-module specification

A modification should be made when testing a multiple-module specification:

- Only those I/O actions that are used to communicate with the outside world should be monitored and synchronized by M.

Therefore, in Figure 11, only those actions that send or receive messages through channels C_1 or C_2 should be changed by the procedure described in Sections 2.1. Other I/O events are considered *internal* and will not be observed.

3 Implementation and Experimental Results

The major advantage of using the transformation approach is to use protocol validation tools for test case generation. A lot of efforts can be saved by using the existing programs. Since the transformation described in this paper is straightforward, the implementation is nearly trivial.

To demonstrate the idea, a protocol validation tool, called *SPIN*, implemented by Holzmann at AT&T is used as the backbone of our test case generator. SPIN uses a C- and CSP-like specification language, called *Promela*, to specify a protocol. The program to be implemented will translate a specification and a fault model written in Promela to a primary P' and a secondary S', and construct a monitor M. The program contains around 1,000 lines of *C++*, *yacc*, and *lex* codes, and is developed by a single person within one week. On a SUN SPARCstation, the program took less than 20 seconds to generate the following test case for the specification in Figure 1 and the fault model in Figure 2.

```
u?Send(1),  R!msg(1,0), u?Send(1), R!msg(1,1), u?Send(1),
R!msg(1,2), R?ack(9),   u?Send(1), R!msg(1,3), u?Send(1)
```

The meaning of the test case is as follows:

1. Input a message Send from channel u. The content of the message is 1.
2. Expect an output, msg(1,0), at channel R, with message 1 and sequence number 0.
3. etc.

If constructing fault models is not desirable, a program that randomly modifies the protocol specification to create fault models is also available. The program has the following capabilities:

1. Randomly modifying the value of a constant (e.g., changing a "4" into a "5").
2. Randomly modifying variable references (e.g., changing a variable x to y).
3. Modifying the operations in an expression (e.g., changing $x + y$ into $x - y$).
4. Altering the control flow (e.g., inserting goto statement arbitrarily).
5. Removing a statement.
6. Combination of the above.

The fault models of the Go-Back-N protocol can automatically be generated by the program described above. The fault model generator can then be integrated with the test case generator to find test cases for random errors.

4 Performance and Complexity Considerations

In order not to confuse with the states in an EFSM, we define a *system state* that is a collection of current states of the modules and current values of the variables. For example, if the sender in Figure 11 is in state S_1, the receiver is in state R_2, the network is in state N_3, and two variables x and y in the protocol specification have values 4 and 5, respectively, the current system state of the protocol will be $\langle S_1, R_2, N_3, x = 4, y = 5 \rangle$.

As shown in Figure 3, P', S' and M are synchronized by the messages passed among them. For example, during an output event, P' sends a message to M and stops until a **resume** signal is received. Similarly, M sends a message to S' and cannot move until an **ok** signal is received. Therefore, automata P', S', and M are synchronized at the point where P issues an output. Between two input or output actions, P', S' and M can run concurrently. Since M does not have any transition to do between two I/O events, the interleaving is between P' and S'.

Since the transitions of P' and S' between two consecutive I/O events are independent, P' and S' can be executed atomically and still yield the same result as if they were executed interleavingly. That is, after a synchronized I/O event, P' can run first until its next I/O event, and then S' can start to run to its next I/O event. Assuming that there are n system states in the protocol specification, and m system states in the fault model, the total number of system states for $\mathcal{I}$ is $O(m+n)$, instead of $O(mn)$. Mostly, m and n are about the same, and the complexity of this method is $O(n)$.

In fact, our implementation has shown good performance, in spite of the inefficiency in using SPIN as a validation tool. To validate a Promela specification, SPIN first generates a C program. Compiling and running the program will generate a path that leads to an error. The path is retraced by SPIN and the I/O events that lead to the deadlock will be recorded by M. It is found that compiling the C program takes most of the execution time. Therefore, if more efficient validation tool were used, the performance would have been even better.

5 Conclusion

Protocol validation problem has been studied for years and many protocol validation tools are available. This paper proposes a method to transform the test case generation problem to a protocol validation problem. Therefore, instead of developing from scratch, a test case generator can be built upon an existing protocol validation tool.

The transformation method takes a protocol specification and a fault model as its input, and generates three automata, namely Primary P', Secondary S', and a Monitor M.

The system containing the three automata is treated as a protocol and fed into a protocol validation tool. A deadlock will occur if there is a discrepancy between the protocol specification and its fault model. By analyzing the path that leads to the deadlock, and carefully record the I/O events, a test case can be found.

The method does not introduce extra complexity to the integrated system $\mathcal{I}$. The total number of system states explored by the protocol validation tool is in about the same order as the total number of system states in the original protocol specification. Therefore, the state explosion problem will not be as serious as a general protocol validation problem.

The method was implemented in a very short period of time. In addition, the experiment showed that the performance of the program is quite acceptable; a test case can be generated within a minute. Therefore, those who have a protocol validation tool may now use this method to transform the validation tool into a test case generator with virtually no extra cost.

References

[1] T. Chow, "Testing software design modeled by finite-state machines," *IEEE Trans. on Software Engineering*, vol. SE-4, no. 3, pp. 178–187, March 1978.

[2] G. Gõnenç, "A model for the design of fault detection experiments," *IEEE Trans. on Computers*, vol. C-19, no. 6, pp. 551–558, June 1970.

[3] S. Naito, "Fault detection for sequential machines by transition tours," in *Proc. 11th IEEE Symp. on Fault Tolerant Computing*, pp. 238–243, 1981.

[4] K. Sabnani and A. Dahbura, "A protocol test generation procedure," *Computer Networks and ISDN Systems*, vol. 15, pp. 285–297, 1988.

[5] G. v. Bochmann and J. Gecsei, "A unified method for the specification and verification of protocols," in *Proc. IFIP Congress '77*, pp. 229–234, 1977.

[6] S. Budkowski and P. Dembinski, "An introduction to Estelle: A specification language for distributed systems," *Computer Networks and ISDN Systems*, vol. 14, no. 1, pp. 3–23, 1987.

[7] T. Bolognesi and E. Brinksma, "Introduction to the ISO specification language LOTOS," *Computer Networks and ISDN Systems*, vol. 14, pp. 25–59, 1987.

[8] E. Brinksma, "A theory for the derivation of tests," in *The Formal Description Technique LOTOS* (P. van Eijk, C.A.Vissers, and M. Diaz, eds.), pp. 235–247, Elsevier Science Publishers B.V. (North-Holland), 1989.

[9] R. Langerak, "A testing theory for LOTOS using deadlock detection," in *Proc. 10th IFIP Symp. on Protocol Specification, Testing, and Verification*, pp. 87–98, 1990.

[10] B. Sarikaya, G. v. Bochmann, and E. Cerny, "A test design methodology for protocol testing," *IEEE Trans. on Software Engineering*, vol. SE-13, no. 5, pp. 518–531, May 1987.

[11] H. Ural, "Test sequence selection based on static data flow analysis," *Computer Communications*, vol. 10, no. 5, pp. 234–242, 1987.

[12] H. Ural and B. Yang, "A test sequence selection method for protocol testing," *IEEE Trans. on Communications*, vol. 39, no. 4, pp. 514–523, April 1991.

[13] C.-J. Wang and M. T. Liu, "Axiomatic test sequence generation for extended finite state machines," in *Proc. 12th International Conference on Distributed Computing Systems*, pp. 252–259, June 1992.

[14] C.-J. Wang and M. T. Liu, "A test suite generation method for extended finite state machines using axiomatic semantics approach," in *IFIP Trans. Protocol Specification, Testing, and Verification, XII*, pp. 29–43, North-Holland, 1992.

[15] G. v. Bochmann, A. Das, R. Dssouli, M. Dubuc, A. Ghedamsi, and G. Luo, "Fault models in testing," in *Protocol Test Systems, IV*, pp. 17–30, Elsevier Science Publisher B.V. (North-Holland), 1992.

[16] C.-J. Wang and M. T. Liu, "Generating test cases for EFSM with given fault models," in *Proc. IEEE INFOCOM '93*, pp. 774–781, March 1993.

[17] C.-J. Wang and M. T. Liu, "Automatic test case generation for Estelle," in *Proc. 1993 Int'l Conf. on Network Protocols*, October 1993.

Protocol Validation Tools as Test Case Generators*

Chang-Jia Wang, Liang-Seng Koh and Ming T. Liu

Department of Computer and Information Science, The Ohio State University,
2036 Neil Ave., Columbus, Ohio 43210-1277, USA

Abstract
A method is proposed to transform test case generation problem to protocol validation problem. Protocol validation has been studied for years and many validation tools are available. By transforming a test case generation problem into a protocol validation problem, a protocol validation tool can be used to generate test cases. The method can be implemented in a very short period of time. The complexity of the proposed method in searching for a test case is $O(n)$, where n is the number of system states in the specification.

1 Introduction

Test case generation (TCG) for *Finite State Machines (FSM)* has been studied for years, and fruitful results have been produced [1, 2, 3, 4]. The FSM is a simple model that ignores the data flow of a protocol and concentrates on the protocol's control flow. In other words, it simplifies a protocol into a state-transition machine. However, ignoring the data flow of the protocol often removes many interesting properties from the protocol. As an example, it is hard to specify a simple protocol like Go-Back-N in FSM without losing those important properties associated with the protocol's window size.

Therefore, most protocol specifications are written in more sophisticated models, such as *Extended Finite State Machine (EFSM)* [5], *Estelle* [6], *LOTOS* [7], or other programming languages. These models are called *extended models* in this paper with respect to the basic FSM model. To be more practical, a test case generation method must be able to generate test cases from extended models. However, the difficulty is that to test an extended model, one needs to test not only the control flow, but also the data flow of the protocol. Testing both control and data flows is considered a challenging problem and

*Research reported herein was supported by U.S. Army Research Office, under contracts No. DAAL03-91-G-0093 and No. DAAL03-92-G-0184. The views, opinions, and/or findings contained in this paper are those of the authors and should not be construed as an official Department of the Army position, policy or decision.

not much work has been done so far. Some references about the test case generation for the extended models can be found in [8, 9, 10, 11, 12, 13, 14].

A test case can be used for detecting a certain type of faults, which are called the *fault models* of the test case [15]. Because of the complexity involved in searching test cases for extended models, it is believed that *fault models* should be used as a guide to simplify the search [16, 17]. Traditionally, a TCG method is dedicated to generating test cases for certain fixed fault models. On the contrary, a TCG method based on fault models takes both a protocol specification and a given fault model as input and generates a test case for the fault model. An I/O sequence can be used as a test case if the specification and the fault model behave differently when applying the I/O sequence.

A fault model is usually a modification of the original protocol specification, and can be described by the same model used to define the protocol specification. For example, Figure 1 is the sender of a Go-Back-N protocol specified in EFSM with a window size equal to four (W=4). A fault model like the one drawn in Figure 2 has a window size being set to five. Note that only the action associated with the first transition is changed[†].

Generating test cases based on a given fault model has the following advantages:

1. Every test case has a well defined *test purpose*. That is, the types of errors a test case can detect is automatically provided. In addition, those who test a protocol implementation with the test case will have the confidence that the fault described by the fault model does not exist in the implementation.
2. It reduces the complexity of finding a test case. Instead of searching aimlessly, a TCG method can use a given fault model as a goal to direct the search.
3. Users can chose to test critical faults first, and test minor errors later.
4. It is more flexible than the traditional methods. The traditional TCG methods have fixed fault models embedded in them. If a critical fault does not contain in the fault models, the traditional TCG methods cannot effectively test this fault.

[†] The EFSMs used in this paper have five types of actions associated with their transitions.

1. Assignment: $W := 4$ assigns value 4 to variable W.
2. Input: $R?Ack(A)$ receives message Ack whose argument is assigned to variable A.
3. Output: $R!Msg(B[Sm], Sm)$ sends a message Msg with arguments $B[Sm]$ and Sm, where $B[Sm]$ is the Sm-th element in array B.
4. Condition: A transition with action $k <> Sm$ can only be executed when k and Sm are not equal.
5. Timeout (T/O): The transition will be executed after a certain amount of time expires.

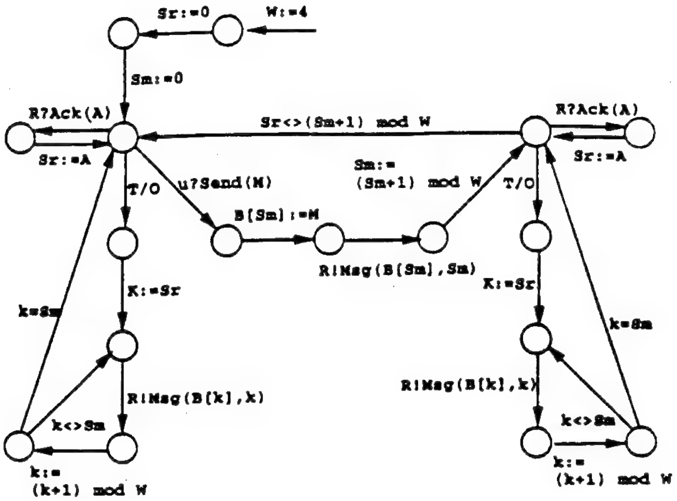


Figure 1: Go-Back-N protocol specified in EFSM

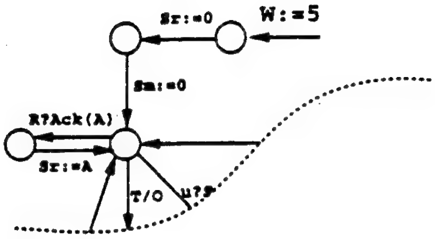


Figure 2: A fault model of Figure 1 (The rest of this figure is the same as Figure 1

In this paper, the problem of TCG based on fault models is transformed into a *protocol validation (PV)* problem. Protocol validation, which is used to detect design errors in a protocol specification, has been studied for years. While many protocol validation tools have been made available, automatic test case generators are still hard to find. Therefore, a method is proposed to transform TCG to PV so that available validation tools can be used to generate test cases. It will also be shown later that the transformation method not only can be used to generate test cases for weak conformance, but also can be used to generate test cases for strong conformance, functional testing, and multiple-module specifications.

It is well known that there is a so called *state explosion* problem for protocol validation, which means that the number of states generated by a protocol validation tool increases exponentially when the protocol specification becomes more complicated. The method proposed in this paper takes a protocol specification and a fault model as its input and generates a combined system to be fed into a protocol validation tool. It is also shown in this paper that the complexity of the combined system is $O(n)$ if the number of global states in the original protocol specification is n. Therefore, the state explosion problem will not occur as long as it will not occur in the original specification.

The organization of this paper is as follows. In Section 2, the transformation from the test case generation problem to the protocol validation problem is proposed. The same section also discusses how the transformation to weak and strong conformance, functional testing, and multiple-module protocols can be applied. In Section 3, the implementation of the test case generator is introduced, and some experimental results are discussed. Finally, the complexity of the test case generator is discussed in Section 4 and a conclusion is given in Section 5.

2 Transforming TCG to PV

2.1 The Transformation

A test case for a fault model is an I/O sequence that detects the discrepancies between the fault model and the protocol specification. That is, the test case is a possible I/O sequence for one, but not for the other. For example, an I/O sequence can be used as a test case if it can be generated from the protocol specification, but cannot be reproduced from the fault model. In other words, if a protocol implementation contains an error described by the fault model, applying the input sequence of the test case to the implementation will produce an unexpected output because the I/O sequence cannot appear in the implementation.

The task is to find an I/O sequence that can only be applied to either the protocol specification or the fault model, but not both. Let P (for *Primary*) be an automaton that describes either the protocol specification or the fault model, and let S (for *Secondary*) be an automaton that describes the other. If P and S are executed synchronously, whenever P makes a certain output, S must make the same output; otherwise, it will be a discrepancy. Similarly, if P requires a certain input, S must wait for the same input; otherwise, it will be a discrepancy as well.

The idea here is to construct a system that will deadlock when P and S take different input or output actions. The combined system can be fed into a protocol validation tool. When the validation tool signals a deadlock error, one can trace through the execution path that causes the deadlock. Then, the I/O events generated by the path can be used as a test case.

The first step is to modify automaton P. Let P' be an automaton the same as P, except for the following differences:

1. For every output action in P, make the same output. Then, wait for a **resume** signal.

2. For every input action in P, output a message that requests the input first. Then, wait for the input.

3. For every timeout in P, output a message labeled `T/O`. Then wait for a **resume** signal.

Next, make an automaton S' similar to S except for the following changes:

1. For every output action in S, input a message first. If the input message is the same as the message to be output, output an ok signal.

2. For every input action in S, input the message. Then, output an ok signal.

3. Remove every timeout in S.

Finally, let M (for *Monitor*) be an automaton communicating with P' and S'. M provides a variable for each parameter of every input action in P'. The variables are initialized to their lower bound values. Monitor M performs the following:

1. Increment the value of a variable by one nondeterministically, unless an upper bound is reached.

2. When an output from P' is received, pass the message to S' and wait for an ok signal from S'. After receiving the ok signal, send a **resume** signal to P', and record the output event.

Table 1: Modification made by P', S', and M

	Original	Primary P'	Secondary S'	Monitor M
Output	$G!msg(x_1, \cdots, x_n)$	$M!msg(x_1, \cdots, x_n)$ $M?resume$	$M?msg(p_1, \cdots, p_n)$ $p_1 = x_1$ $\vdots$ $p_n = x_n$ $M!ok$	$P'?msg(p_1, \cdots, p_n)$ $S'!msg(p_1, \cdots, p_n)$ $S'?ok$ $P'!resume$
Input	$G?msg(x_1, \cdots, x_n)$	$M!request(msg)$ $M?msg(x_1, \cdots, x_n)$	$M?msg(x_1, \cdots, x_n)$ $M!ok$	$P'?request(msg)$ $S'!msg(x_1, \cdots, x_n)$ $S'?ok$ $P'!msg(x_1, \cdots, x_n)$
Timeout	T/O	$M!T/O$ $M?resume$		$P'?T/O$ $P'!resume$

3. When P' asks for an input message, send the message to S' first. After receiving a ok from S', send the same message to P' and record the input event.

4. When P' outputs a `T/O` message, send **resume** signals to P', and record the timeou event.

The above modification is summarized in Table 1. Let $\mathcal{I}$ (for *Integrated*) be an integrated system that contains P', S', and M. Then, $\mathcal{I}$ will deadlock if P generates som I/O sequences that cannot be reproduced by S. This can be briefly proved as follows.

1. If S cannot generate the same message output by P, S' will stop. Without receiving any **ok** signal, M will be blocked and will not send the **resume** signal to P'. Therefore, P' will be blocked as well.

2. If S will not receive the same input as P does, S' will be blocked and no ok signal will be send. Therefore, M will be blocked and P' will never receive the **resume** signal. The system deadlocks.

The protocol among M, P' and S' is illustrated in Figure 3. When P' sends a message **msg** to M, M passes it to S'. If the **msg** is also a message that S' intends to output, S' will send an **ok**. Upon receiving the **ok** signal, M sends a **resume** to P' (Figure 3a). On the other hand, if P' needs to receive a certain message, it sends a request to M. Upon receiving the request, M sends the message to S' first. If an **ok** is received, M sends the requested message to P' (Figure 3b). The timeout message between P' and M' are used to record the timeout event, and will not affect the overall execution of the system.

An example of the transformation is shown in Figures 4 to 6. Figure 4 is the primary automaton P' modified from the Go-Back-N protocol specification in Figure 1. The

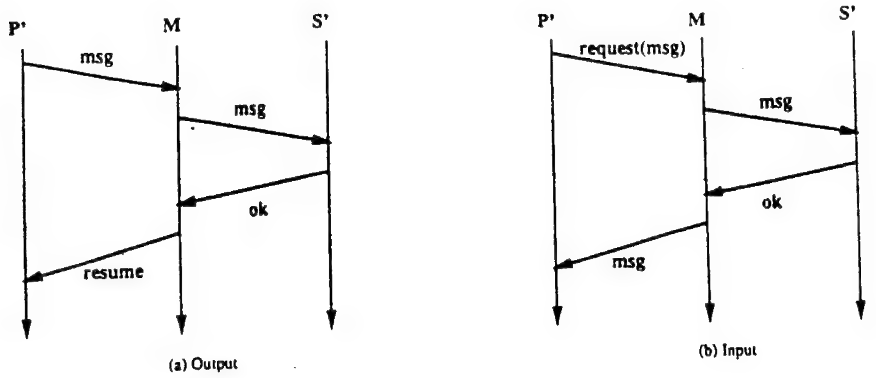


Figure 3: Communication among M, P' and S'

shaded arrows and circles are the transitions and states being changed according to the rules described in Table 1. Figure 5 is the secondary automaton S' modified from the fault model shown in Figure 2. Monitor M for Figures 4 and 5 is drawn in Figure 6. Note that all the I/O events in P' and S' are changed into communication between P' and M, and between S' and M, respectively.

When a protocol specification and its fault models are specified by a formal model (such as FSM, EFSM, Estelle, LOTOS, SDL, Petri-Net, Promela, etc.), it is trivial to transform P and S into P' and S', respectively. It is also straightforward to construct the monitor M. Automata P', S' and M can be fed into a validation tool for the formal model and a deadlock can be found if there is an I/O sequence that is possible for P but cannot be generated by S. By retracing the path leading to the deadlock, monitor M will record the I/O events along the path. The recorded I/O event sequence can then be used as a test case.

2.2 Weak and Strong Conformance

There are two types of conformance between a protocol specification and its implementation. The *weak conformance* indicates that a protocol implementation should perform every function defined in its specification. In other words, things that ought to happen should happen. The *strong conformance* imposes an additional constrain which states that a protocol implementation cannot have any behavior not defined by the protocol specification. That is, things that cannot happen must not occur.

A fault model violating the weak conformance cannot perform some functions defined by the protocol specification. In other words, there exists an I/O event sequence that

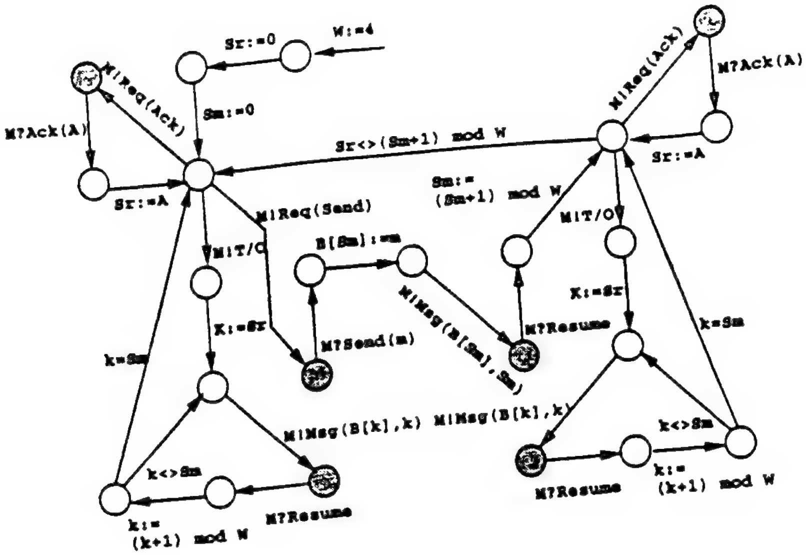


Figure 4: Modified protocol specification P'

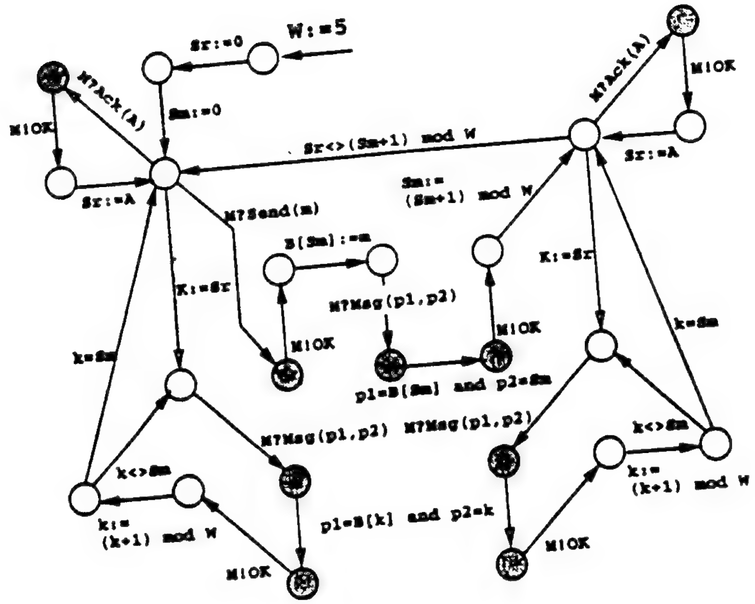


Figure 5: Modified fault model S'

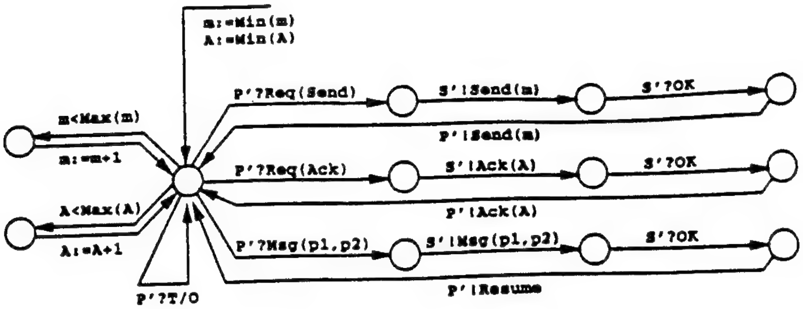


Figure 6: Monitor M

can be generated by the specification, but cannot happen in the fault model. To generate a test case for the fault model, one can use the protocol specification as the primary automaton P, use the fault model as the secondary automaton S, apply the transformation in Section 2.1, and feed the resulting system $\mathcal{I}$ into a protocol validation tool.

A fault model that violates the strong conformance has some execution paths that are not specified in the protocol specification. That is, some I/O sequence generated by the fault model can never happen in the specification. Thus, if one use the fault model as the primary automaton P and use the specification as the secondary automaton S, one can find a test case with the method described in Section 2.1. When the input of the test case is applied to a protocol implementation, and an expected output appears, the implementation must contain an error described by the fault model.

It is possible that a fault model violates both weak and strong conformance. In this case, treating the protocol specification as either the primary or the secondary will work. Sometimes, however, it is difficult to tell whether a fault model violates the weak or strong conformance. In such a case, if one method cannot find a test case, the other must be tried.

2.3 Functional Testing

It is useful to create test cases that ensure the correctness of certain behavior in the protocol specification. Such "behavior" is often called a *function* or a *requirement* of the protocol specification. For example, a requirement of the Go-Back-N protocol in Figure 1 can be "*there are no more than three messages in the channel at any given time.*"

In order to test if a protocol implementation conforms with the requirement, a fault model that violates the requirement is constructed. However, since a requirement usually is just a small portion of the original protocol specification, it will be quite troublesome to construct a fault model from the entire protocol specification. In fact, a fault model can be created by including only those transitions necessary to violate the requirement. For example, a fault model violating the requirement described in the above paragraph can be drown as an EFSM in Figure 7. In the figure, there is no restriction on how many messages can be sent to the receiver consecutively before the sender receives an acknowledgment.

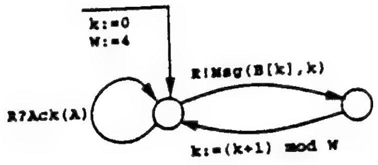


Figure 7: A fault model that sends more than three messages at a time.

The fault model in Figure 7 defines some behavior that should not exist in a correct implementation, so it violates the strong conformance requirement. Therefore, it is used as a primary automaton and the specification is used as a secondary. Since the fault model only specifies partial behavior of the entire specification, some I/O events in the specification may not exist in the fault model. Hence, a modification of the transformation described in Section 2.1 should be noted as follows.

- To change P (S) into P' (S'):
  1. If an input or output action exists in both P and S, follow the procedure in Section 2.1.
  2. For every output message that exists only in P (S), send the output to M and wait for a **resume** signal from M.
  3. For every input message that exists only in P (S), send an request for the input to M and wait for the input event from M.
- To construct M:
  1. For those input or output actions that exist in both P and S, follow the procedure in Section 2.1.
  2. If an output message that exists only in P (S) is received from P' (S'), record the output event and send a **resume** signal to P' (S').
  3. If a request for input that exists only in P (S) is received from P' (S'), record the input event and send the input message to P' (S').

In other words, if an input or output message exists only in P or S, the monitor M will not try to synchronize P' and S' with the message. It will only record the I/O event and release the automaton that generates the event. For example, in Figure 1, the input event u?Send(m) exists only in the specification, and should be modified according to the above procedure. The resulting EFSMs P', S', and M are shown in Figures 8, 9, and 10, respectively. Note that the shaded nodes and arrows in Figures 9 and 1 show the difference between the method in this section and the one in Section 2.1.

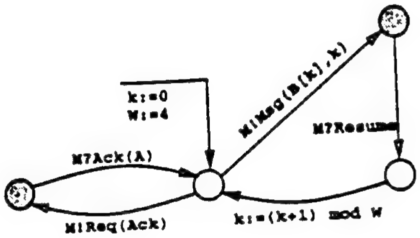


Figure 8: The primary P' from the fault model in Figure 7.

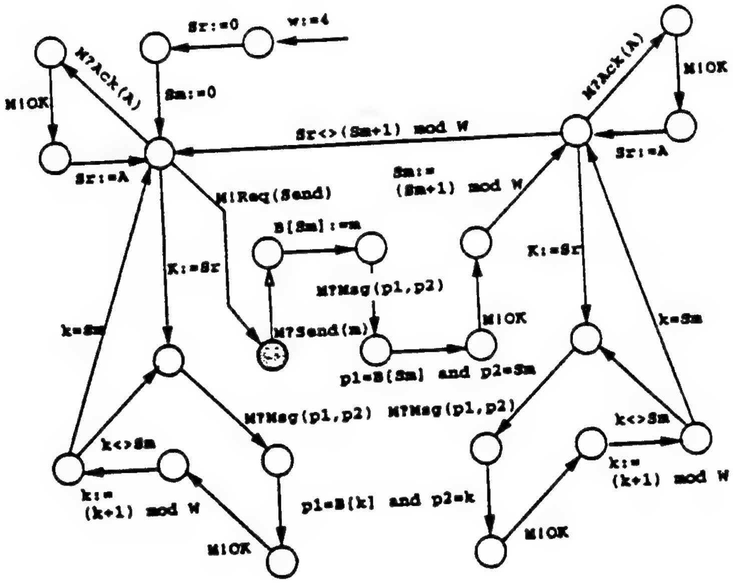


Figure 9: The secondary S' from the specification in Figure 1.

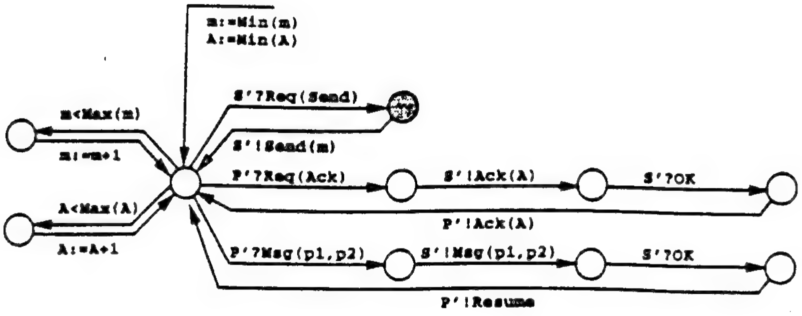


Figure 10: The monitor M from the fault model in Figure 7 and the specification in Figure 1.

2.4 Test Case Generation for Multiple Modules

Test cases can also be generated for multiple modules using this method. For example, one may want to test how a protocol containing a sender and a receiver operates when connected with a network. The whole system can be modeled like the one in Figure 11.

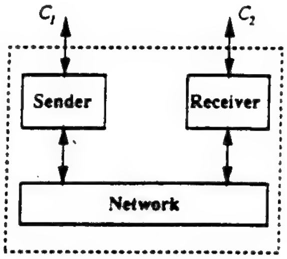


Figure 11: A multiple-module specification

A modification should be made when testing a multiple-module specification:

- Only those I/O actions that are used to communicate with the outside world should be monitored and synchronized by M.

Therefore, in Figure 11, only those actions that send or receive messages through channels C_1 or C_2 should be changed by the procedure described in Sections 2.1. Other I/O events are considered *internal* and will not be observed.

3 Implementation and Experimental Results

The major advantage of using the transformation approach is to use protocol validation tools for test case generation. A lot of efforts can be saved by using the existing programs. Since the transformation described in this paper is straightforward, the implementation is nearly trivial.

To demonstrate the idea, a protocol validation tool, called *SPIN*, implemented by Holzmann at AT&T is used as the backbone of our test case generator. SPIN uses a C- and CSP-like specification language, called *Promela*, to specify a protocol. The program to be implemented will translate a specification and a fault model written in Promela to a primary P' and a secondary S', and construct a monitor M. The program contains around 1,000 lines of *C++*, *yacc*, and *lex* codes, and is developed by a single person within one week. On a SUN SPARCstation, the program took less than 20 seconds to generate the following test case for the specification in Figure 1 and the fault model in Figure 2.

```
u?Send(1),  R!msg(1,0), u?Send(1), R!msg(1,1), u?Send(1),
R!msg(1,2), R?ack(9),   u?Send(1), R!msg(1,3), u?Send(1)
```

The meaning of the test case is as follows:

1. Input a message `Send` from channel `u`. The content of the message is `1`.
2. Expect an output, `msg(1,0)`, at channel `R`, with message `1` and sequence number `0`.
3. etc.

If constructing fault models is not desirable, a program that randomly modifies the protocol specification to create fault models is also available. The program has the following capabilities:

1. Randomly modifying the value of a constant (e.g., changing a "4" into a "5").
2. Randomly modifying variable references (e.g., changing a variable x to y).
3. Modifying the operations in an expression (e.g., changing $x + y$ into $x - y$).
4. Altering the control flow (e.g., inserting `goto` statement arbitrarily).
5. Removing a statement.
6. Combination of the above.

The fault models of the Go-Back-N protocol can automatically be generated by the program described above. The fault model generator can then be integrated with the test case generator to find test cases for random errors.

4 Performance and Complexity Considerations

In order not to confuse with the states in an EFSM, we define a *system state* that is a collection of current states of the modules and current values of the variables. For example, if the sender in Figure 11 is in state S_1, the receiver is in state R_2, the network is in state N_3, and two variables x and y in the protocol specification have values 4 and 5, respectively, the current system state of the protocol will be $\langle S_1, R_2, N_3, x = 4, y = 5 \rangle$.

As shown in Figure 3, P', S' and M are synchronized by the messages passed among them. For example, during an output event, P' sends a message to M and stops until a resume signal is received. Similarly, M sends a message to S' and cannot move until an ok signal is received. Therefore, automata P', S', and M are synchronized at the point where P issues an output. Between two input or output actions, P', S' and M can run concurrently. Since M does not have any transition to do between two I/O events, the interleaving is between P' and S'.

Since the transitions of P' and S' between two consecutive I/O events are independent, P' and S' can be executed atomically and still yield the same result as if they were executed interleavingly. That is, after a synchronized I/O event, P' can run first until its next I/O event, and then S' can start to run to its next I/O event. Assuming that there are n system states in the protocol specification, and m system states in the fault model, the total number of system states for $\mathcal{I}$ is $O(m+n)$, instead of $O(mn)$. Mostly, m and n are about the same, and the complexity of this method is $O(n)$.

In fact, our implementation has shown good performance, in spite of the inefficiency in using SPIN as a validation tool. To validate a Promela specification, SPIN first generates a C program. Compiling and running the program will generate a path that leads to an error. The path is retraced by SPIN and the I/O events that lead to the deadlock will be recorded by M. It is found that compiling the C program takes most of the execution time. Therefore, if more efficient validation tool were used, the performance would have been even better.

5 Conclusion

Protocol validation problem has been studied for years and many protocol validation tools are available. This paper proposes a method to transform the test case generation problem to a protocol validation problem. Therefore, instead of developing from scratch, a test case generator can be built upon an existing protocol validation tool.

The transformation method takes a protocol specification and a fault model as its input, and generates three automata, namely Primary P', Secondary S', and a Monitor M.

The system containing the three automata is treated as a protocol and fed into a protocol validation tool. A deadlock will occur if there is a discrepancy between the protocol specification and its fault model. By analyzing the path that leads to the deadlock, and carefully record the I/O events, a test case can be found.

The method does not introduce extra complexity to the integrated system $\mathcal{I}$. The total number of system states explored by the protocol validation tool is in about the same order as the total number of system states in the original protocol specification. Therefore, the state explosion problem will not be as serious as a general protocol validation problem.

The method was implemented in a very short period of time. In addition, the experiment showed that the performance of the program is quite acceptable; a test case can be generated within a minute. Therefore, those who have a protocol validation tool may now use this method to transform the validation tool into a test case generator with virtually no extra cost.

References

[1] T. Chow, "Testing software design modeled by finite-state machines," *IEEE Trans. on Software Engineering*, vol. SE-4, no. 3, pp. 178–187, March 1978.

[2] G. Gõnenç, "A model for the design of fault detection experiments," *IEEE Trans. on Computers*, vol. C-19, no. 6, pp. 551–558, June 1970.

[3] S. Naito, "Fault detection for sequential machines by transition tours," in *Proc. 11th IEEE Symp. on Fault Tolerant Computing*, pp. 238–243, 1981.

[4] K. Sabnani and A. Dahbura, "A protocol test generation procedure," *Computer Networks and ISDN Systems*, vol. 15, pp. 285–297, 1988.

[5] G. v. Bochmann and J. Gecsei, "A unified method for the specification and verification of protocols," in *Proc. IFIP Congress '77*, pp. 229–234, 1977.

[6] S. Budkowski and P. Dembinski, "An introduction to Estelle: A specification language for distributed systems," *Computer Networks and ISDN Systems*, vol. 14, no. 1, pp. 3-23, 1987.

[7] T. Bolognesi and E. Brinksma, "Introduction to the ISO specification language LOTOS," *Computer Networks and ISDN Systems*, vol. 14, pp. 25–59, 1987.

[8] E. Brinksma, "A theory for the derivation of tests," in *The Formal Description Technique LOTOS* (P. van Eijk, C.A.Vissers, and M. Diaz, eds.), pp. 235–247, Elsevier Science Publishers B.V. (North-Holland), 1989.

[9] R. Langerak, "A testing theory for LOTOS using deadlock detection," in *Proc. 10th IFIP Symp. on Protocol Specification, Testing, and Verification*, pp. 87–98, 1990.

[10] B. Sarikaya, G. v. Bochmann, and E. Cerny, "A test design methodology for protocol testing," *IEEE Trans. on Software Engineering*, vol. SE-13, no. 5, pp. 518–531, May 1987.

[11] H. Ural, "Test sequence selection based on static data flow analysis," *Computer Communications*, vol. 10, no. 5, pp. 234–242, 1987.

[12] H. Ural and B. Yang, "A test sequence selection method for protocol testing," *IEEE Trans. on Communications*, vol. 39, no. 4, pp. 514–523, April 1991.

[13] C.-J. Wang and M. T. Liu, "Axiomatic test sequence generation for extended finite state machines," in *Proc. 12th International Conference on Distributed Computing Systems*, pp. 252–259, June 1992.

[14] C.-J. Wang and M. T. Liu, "A test suite generation method for extended finite state machines using axiomatic semantics approach," in *IFIP Trans. Protocol Specification, Testing, and Verification, XII*, pp. 29–43, North-Holland, 1992.

[15] G. v. Bochmann, A. Das, R. Dssouli, M. Dubuc, A. Ghedamsi, and G. Luo, "Fault models in testing," in *Protocol Test Systems, IV*, pp. 17–30, Elsevier Science Publisher B.V. (North-Holland), 1992.

[16] C.-J. Wang and M. T. Liu, "Generating test cases for EFSM with given fault models," in *Proc. IEEE INFOCOM '93*, pp. 774–781, March 1993.

[17] C.-J. Wang and M. T. Liu, "Automatic test case generation for Estelle," in *Proc. 1993 Int'l Conf. on Network Protocols*, October 1993.